Q4)

Repeat the same analysis in Q2 and Q3 with sparse inverse covariance. Discuss how the results differ from using the correlation matrix for connectivity analysis.

In [5]:

```python
# Import the requred Packages
import os
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import nibabel as nib

fmri_file = '../../datasets/fMRI/HW4/sub-001_ses-001_task-eoec_bold.nii.gz' # Get the nifti file
img = nib.load(fmri_file) # Load in the nifti file
print(type(img)) # Print the type of the img variable (should be nifti)
print(img.shape) # Print the dimensions of the object
```

```
<class 'nibabel.nifti1.Nifti1Image'>
(64, 64, 35, 120)
```

Import Masker & Atlas

To get the ROI time series data from the whole fMRI dataset.

In [6]:

```python
# Load the required Packages
import nilearn as nl
import numpy as np
# Retrieve the atlas and the data
from nilearn import datasets
# Fetch the atlas file.
atlas = datasets.fetch_atlas_msdl()
# Loading the the Probabilistic atlas image
atlas_filename = atlas['maps']
# Loading the list containing the labels of the regions
labels = atlas['labels']
# Extract time series
data = img.get_fdata() # Get the 4 dimentional data from the fMRI
# import maskers
from nilearn.maskers import NiftiMapsMasker
masker = NiftiMapsMasker(maps_img=atlas_filename, standardize=True, memory='nilearn_cache', verbose=5) # get ma
time_series = masker.fit_transform(img) # get time series from fMRI fitted with the given masker
```

```
[NiftiMapsMasker.wrapped] loading regions from None
Resampling maps
[Memory]0.0s, 0.0min    : Loading resample_img...
________________________________________resample_img cache loaded - 0.0s, 0.0min
/home/joshua/.local/lib/python3.10/site-packages/nilearn/maskers/base_masker.py:253: UserWarning: memory_level i
s currently set to 0 but a Memory object has been provided. Setting memory_level to 1.
  return self.transform_single_imgs(
[Memory]0.3s, 0.0min    : Loading _filter_and_extract...
__________________________________filter_and_extract cache loaded - 0.0s, 0.0min
```

Split EO (Eyes Open) & EC (Eyes Closed) data

into 2 different time series arrays at all ROIs.

In [7]:

```python
# Timing table (onset, duration, trial_type) fitted to the timed intervals (120)
timing_info = [
    (0, 10, "EC"), (10, 10, "EO"), (20, 10, "EC"), (30, 10, "EO"),
    (40, 10, "EC"), (50, 10, "EO"), (60, 10, "EC"), (70, 10, "EO"),
    (80, 10, "EC"), (90, 10, "EO"), (100, 10, "EC"), (110, 10, "EO")
]

# Initialize masks
ec_mask = np.zeros(120, dtype=bool)  # For EC condition
eo_mask = np.zeros(120, dtype=bool)  # For EO condition

# Create masking indecies for EC and EO
for onset, duration, trial_type in timing_info:
    if trial_type == "EC":
        ec_mask[onset:onset + duration] = True
    elif trial_type == "EO":
        eo_mask[onset:onset + duration] = True

# Apply masks to the 4D fMRI data
ec_data = data[..., ec_mask]  # EC condition data
```

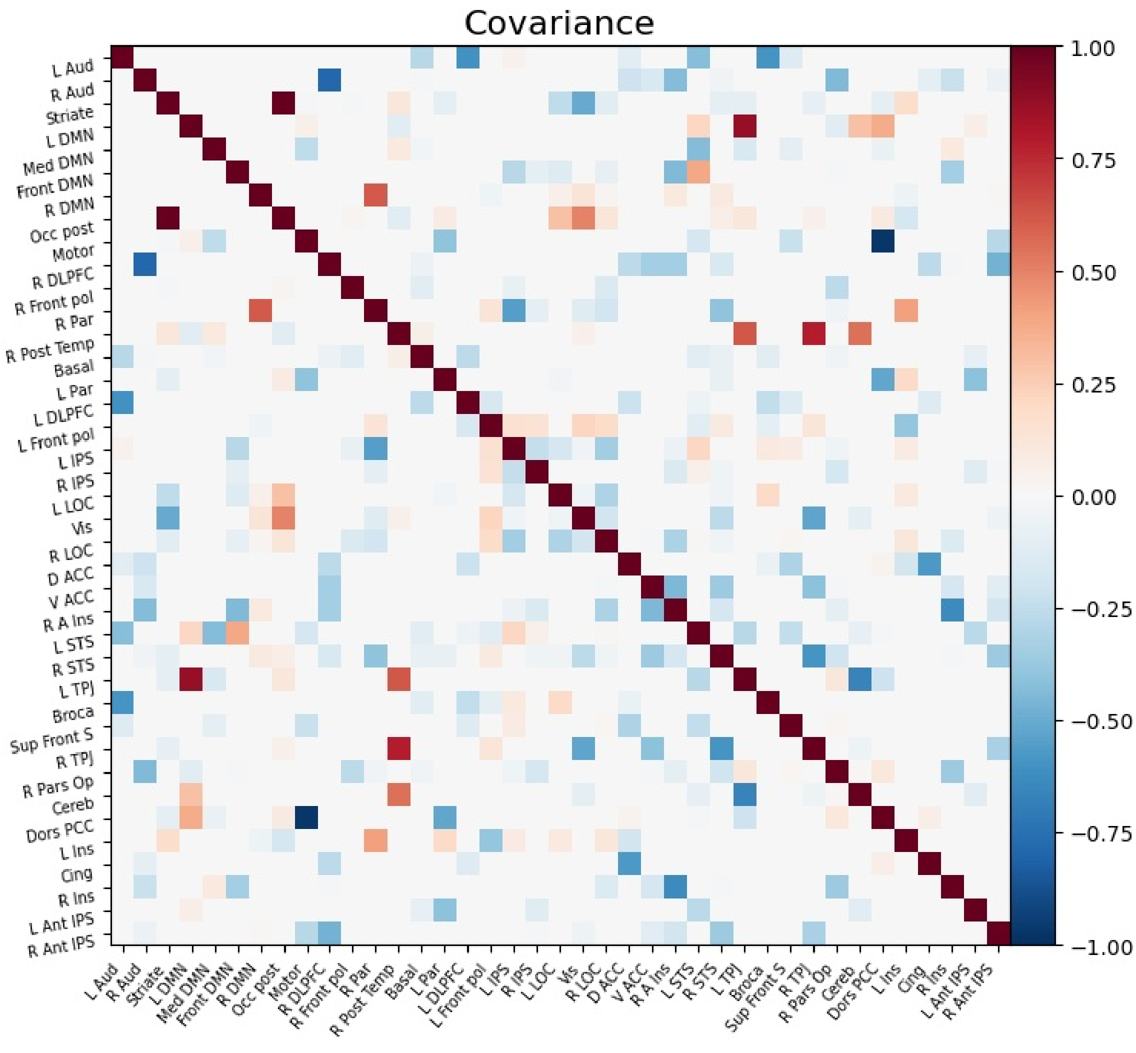
```
eo_data = data[..., eo_mask]  # EO condition data

# Get random ROI (region of interest)
roi_time_series = time_series[:]  # Full time series for the all ROI
roi_ec_time_series = roi_time_series[ec_mask]  # Time series for EC condition
roi_eo_time_series = roi_time_series[eo_mask]  # Time series for EO condition
```

Plot EO (eyes open) Sparce Inverse Covariance Matrix

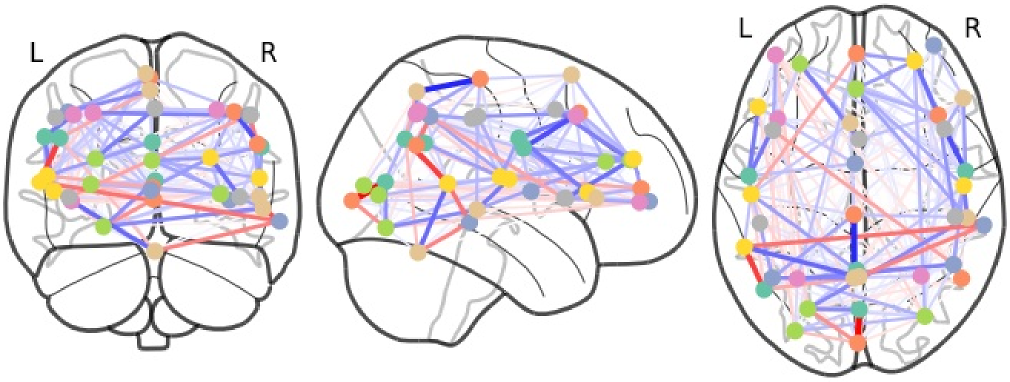
In [10]:
```
# from nilearn import plotting
from nilearn.connectome import ConnectivityMeasure

# Display the connectome matrix
from nilearn import plotting

# Import lasso for inverse covariance
from sklearn.covariance import GraphicalLassoCV
graphical_lasso = GraphicalLassoCV() # create lasso object
graphical_lasso.fit(roi_eo_time_series) # fit the lasso object to the time series data
inverse_matrix = graphical_lasso.precision_ # get the inverse matrix from the fitted lasso

# Display the covariance
plotting.plot_matrix(inverse_matrix, labels=labels, figure=(9,7), vmax=1, vmin=-1, title='Covariance')

coords = atlas.region_coords # get coords in all ROIs from atlas

# Display the corresponding brain connectivity
plotting.plot_connectome(inverse_matrix, coords,title='Covariance')
```

```
/home/joshua/.local/lib/python3.10/site-packages/numpy/core/_methods.py:173: RuntimeWarning: invalid value encou
ntered in subtract
  x = asanyarray(arr - arrmean)
/home/joshua/.local/lib/python3.10/site-packages/sklearn/covariance/_graph_lasso.py:184: ConvergenceWarning: gra
phical_lasso: did not converge after 100 iteration: dual gap: -3.899e-04
  warnings.warn(
```

Out[10]: `<nilearn.plotting.displays._projectors.OrthoProjector at 0x772db5359840>`



Covariance



EO Inverse Covariance Plot Analysis

Looking at the plot, there seems to be a lot of negative corraltions compared to the corralation matrix and this makes since due to us analysing the inverse corralation matrix. However the corralations on both sides are weak, I would say that on average the negative corralations are stronger, but all in all relatively low corralation in most regions. There are some stronger corrlations near the back (occipital lobe) and the front (frontal lobe), but only for a very select connections. This is in stark contrast in comparision to the corralation matrix from Q2 which had lots of strong corralation connections.

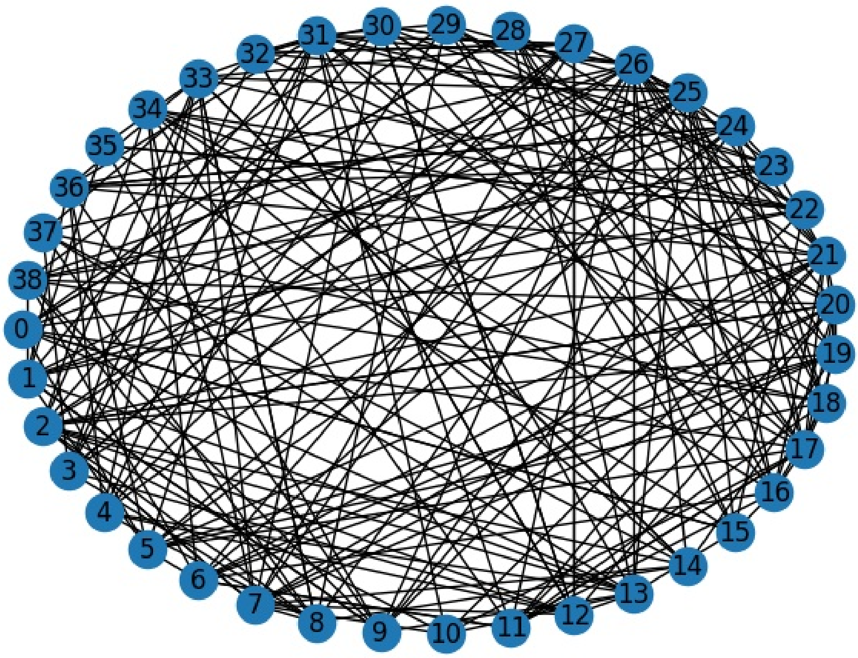
Local Graph Measures EO (eyes open)

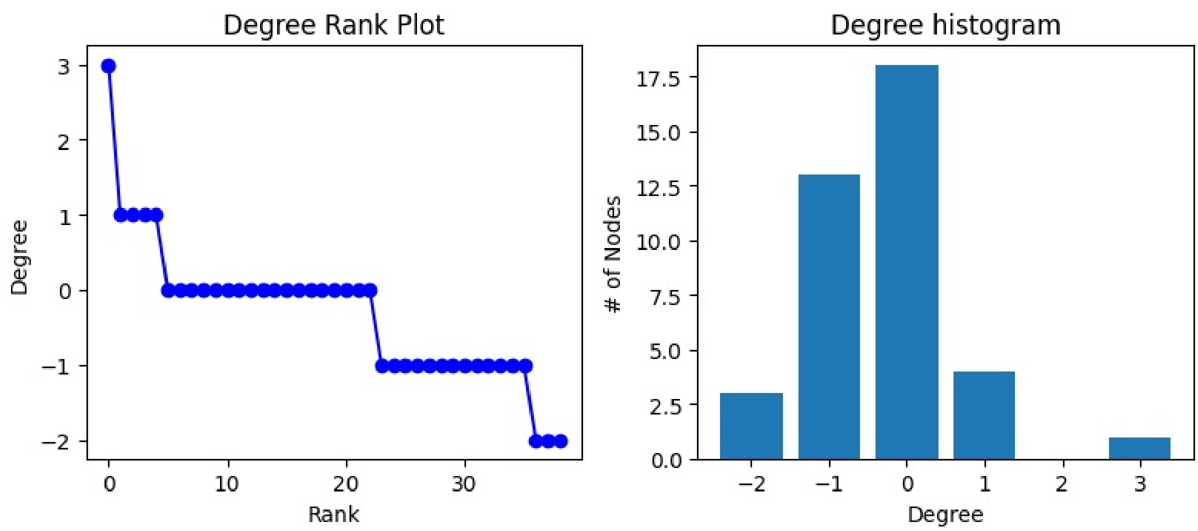
In [12]:

```
# Import networkx for graph use
import networkx as nx

# from nilearn import plotting
from nilearn.connectome import ConnectivityMeasure

# Display the connectome matrix
from nilearn import plotting

covariance_matrix = inverse_matrix
covariance_matrix = np.array(covariance_matrix) # convert covariance matrix into numpy array

np.fill_diagonal(covariance_matrix, 0) # make sure there are no connections to the node's self

# Convert to NetworkX graphs
graph = nx.from_numpy_array(covariance_matrix)

# Iterate over the graph
for i, j in graph.edges():
    graph[i][j]['weight'] = round(covariance_matrix[i, j]) # round the weights to be a whole number for analysi

clustering_coefficient = nx.clustering(graph, weight='weight') # calculating the cluster coef
print("Clustering Coefficients:", clustering_coefficient) # display the cluster coef
# Get the minimum and maximum coefficients
min_coef = min(clustering_coefficient.values())
max_coef = max(clustering_coefficient.values())
print("Min coef", min_coef)
print("Max coef", max_coef)

widths = nx.get_edge_attributes(graph, 'weight') # get graph edge attributes
pos = nx.shell_layout(graph) # get the positions of the graph
nx.draw(graph,pos,with_labels=True) # draw the graph nodes
nx.draw_networkx_edges(graph, pos,edgelist=widths.keys(),width=list(widths.values())) # draw the graph edges

# Plot the degree distribution in the graph
degree_sequence = sorted((d for n, d in graph.degree(weight='weight')), reverse=True) # get weighted degree dis
dmin = min(degree_sequence) # get the min weighted degree value
dmax = max(degree_sequence) # get the max weighted degree value
fig = plt.figure(figsize=(8, 8)) # make the plot easily visible
axgrid = fig.add_gridspec(5, 4) # add axis grid
ax1 = fig.add_subplot(axgrid[3:, :2]) # make the first plot
```

```
ax1.plot(degree_sequence, "b-", marker="o") # add normal plot for the degree weights
ax1.set_title("Degree Rank Plot") # set the title
ax1.set_ylabel("Degree") # set the y-axis
ax1.set_xlabel("Rank") # set the x-axis
ax2 = fig.add_subplot(axgrid[3:, 2:]) # make the second plot
ax2.bar(*np.unique(degree_sequence, return_counts=True)) # add bar plot the the degree weights
ax2.set_title("Degree histogram") # set the title
ax2.set_xlabel("Degree") # set the y-axis
ax2.set_ylabel("# of Nodes") # set the x-axis
fig.tight_layout() # set the layout for easier visuals
plt.show() # display plot
```

```
Clustering Coefficients: {0: 0, 1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0, 8: 0, 9: 0, 10: 0, 11: 0, 12: -0.02777
7777777777776, 13: 0, 14: 0, 15: 0, 16: 0, 17: 0, 18: 0, 19: 0, 20: 0, 21: 0, 22: 0, 23: 0, 24: 0, 25: 0, 26: 0,
27: -0.027777777777777776, 28: 0, 29: 0, 30: 0, 31: 0, 32: -0.03571428571428571, 33: 0, 34: 0, 35: 0, 36: 0, 37:
0, 38: 0}
Min coef -0.03571428571428571
Max coef 0
```





Local Graph Measures EO Analysis

(1) The Clustering Coefficients are ranging from -0.035714 to 0, where 0 is for clusters with low corralation to one another and -0.035714 is for the higher weighted clusters with high (negative) corralation to one another. This is different from Q3 due to the minmum there being 0 and the maximum here being 0, showing us that the clustering coefficients here are dealing with more negative values.

(2) Each of the nodes are verying in terms of connectivity some are well connected with 10+ edges, other are not with ~5 edges.Previously all nodes in Q3 seemed to all be connected to one another directly due to the density of the lines in the graph image.

(3.a) The Degree Rank Plot shows the the degree weight along with the rank of that node. Since there are 39 nodes there should roughly be 39 ranks and we can see that from the plot. Values of the weighted degrees are ranging from -2 to 3 which is 5 points in separation from the highest to lowest weith. This is 15 points less in comparision to EO Q3 (total 20 points) & meaning that the corralation matrix had 4x more range in terms of corralation strength vs the inverse sparce matrix.

(3.b) The Degree historgram shows the degree weight along with the number of nodes which is similar to the degree rank plot, but shows us a better picuture on the frequncy of each weighted degree. Showing us that degree 0 had the highest frequency rate at 18, telling us that there are a lot of no corralated nodes in the graph and this number is way higher that Q3 which was 6 at the highest.

Global Graph Measures EO (eyes open)

In [13]:
```
global_efficiency = nx.global_efficiency(graph)
average_clustering = nx.average_clustering(graph, weight='weight')

print(global_efficiency)
print(average_clustering)
print(nx.diameter(graph))
```

```
0.6264057579847068
-0.0023402523402523403
3
```

Global Graph Measures EO Analysis

(1) The Global efficiency of the graph is 0.626 which tells us that this is a relativly well connected graph considering the scale from 0 to 1, however this number is less than the perfectly connected graph we had in the corralation matrix case.

(2) The average cluster of the graph is -0.00234 which tells what the average correlation of a cluster roughly is. This is a way smaller number than we had previously at 0.03348 and that means there is way less corraltion going on in the inverse sparce matrix in comparision to the corralation matrix.

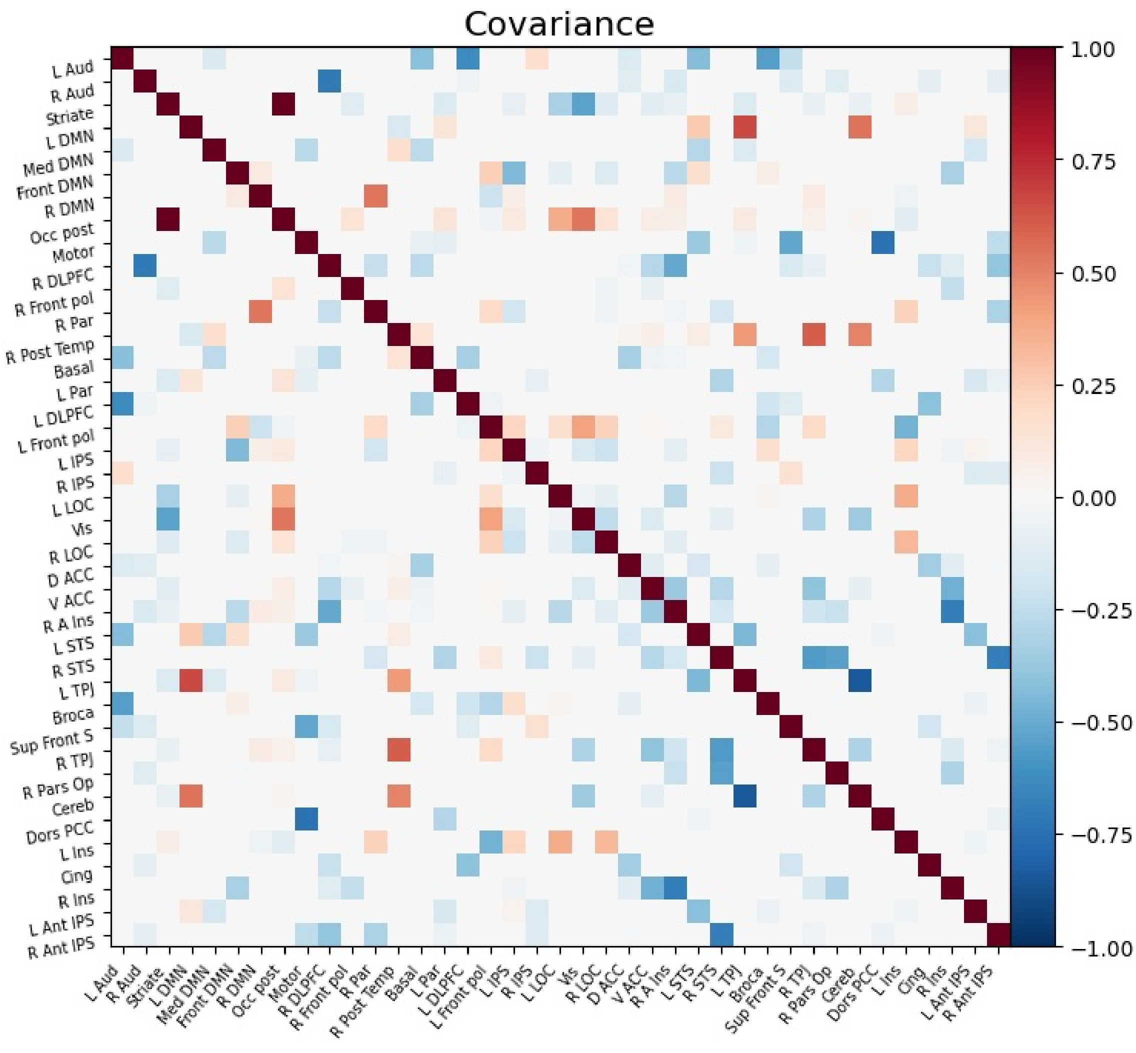
(3) The diameter of the graph is 3 meaning the longest shortest path between any two nodes is 3. This is longer than the 1 we had in Q3 and this makes since because we don't have a perfectly connected graph like in the corralation matrix case.

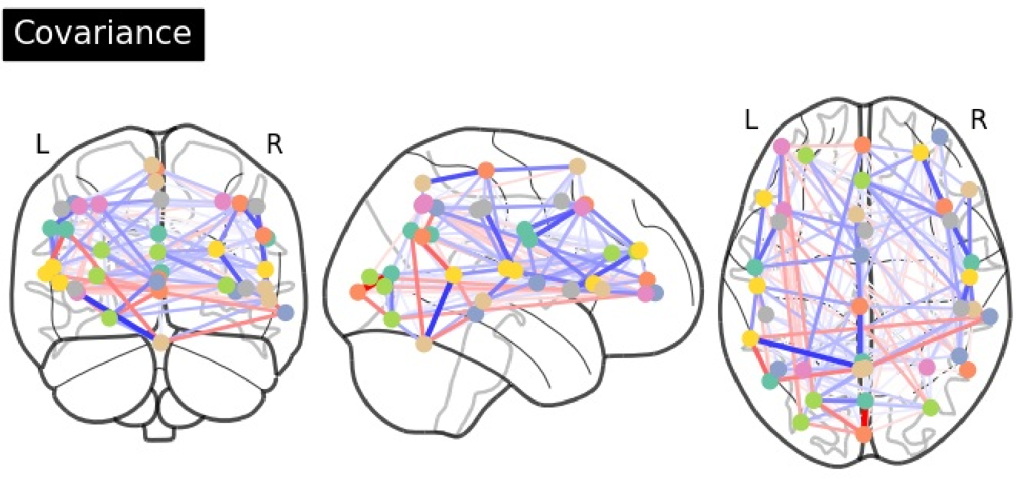
Plot EC (eyes closed) Sparce Inverse Covariance Matrix

In [14]:
```
# from nilearn import plotting
from nilearn.connectome import ConnectivityMeasure

# Display the connectome matrix
from nilearn import plotting

# Import lasso for inverse covariance
from sklearn.covariance import GraphicalLassoCV
graphical_lasso = GraphicalLassoCV() # create lasso object
graphical_lasso.fit(roi_ec_time_series) # fit the lasso object to the time series data
inverse_matrix = graphical_lasso.precision_ # get the inverse matrix from the fitted lasso

# Display the covariance
plotting.plot_matrix(inverse_matrix, labels=labels, figure=(9,7), vmax=1, vmin=-1, title='Covariance')

coords = atlas.region_coords # get coords in all ROIs from atlas

# Display the corresponding brain connectivity
plotting.plot_connectome(inverse_matrix, coords,title='Covariance')
```

```
/home/joshua/.local/lib/python3.10/site-packages/numpy/core/_methods.py:173: RuntimeWarning: invalid value encou
ntered in subtract
  x = asanyarray(arr - armean)
```

Out[14]:
```
<nilearn.plotting.displays._projectors.OrthoProjector at 0x772dadcffa90>
```





EC Inverse Covariance Plot Analysis

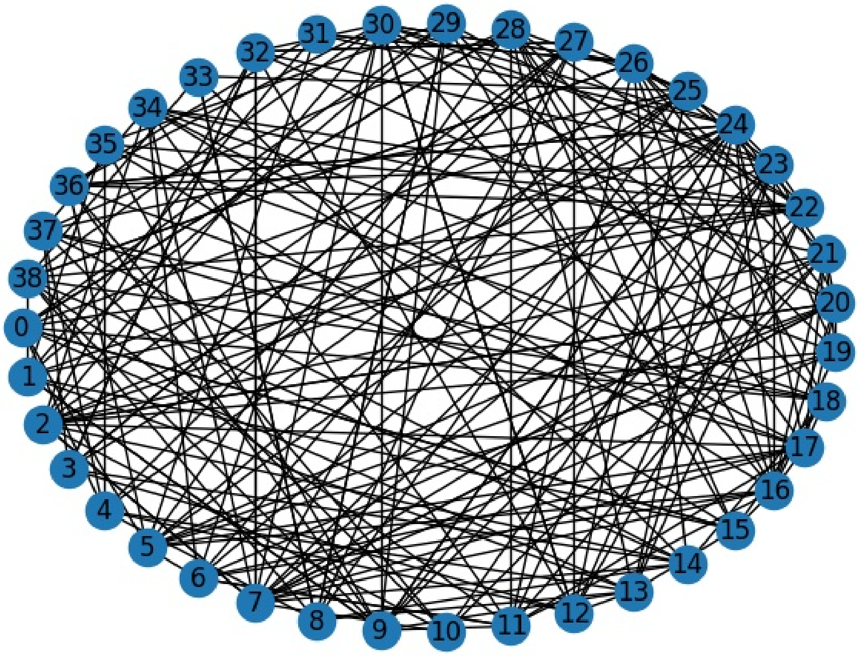
Looking at the plot, there seems to be a lot of negative corraltions compared to the corralation matrix and this makes since due to us analysing the inverse corralation matrix. However the corralations on both sides are weak, I would say that on average the negative corralations are stronger, but all in all relatively low corralation in most regions. There are some stronger corrlations near the back (occipital lobe) and the front (frontal lobe), but only for a very select connections. This is in stark contrast in comparision to the corralation matrix from Q2 which had lots of strong corralation connections.

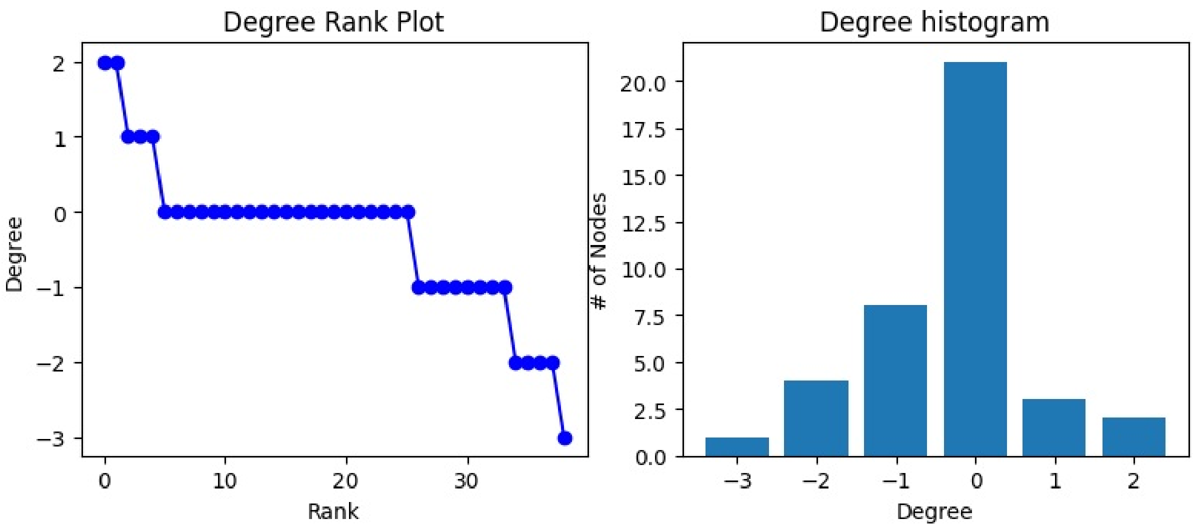
Local Graph Measures EC (eyes closed)

In [15]:

```
# Import networkx for graph use
import networkx as nx

# from nilearn import plotting
from nilearn.connectome import ConnectivityMeasure

# Display the connectome matrix
from nilearn import plotting

covariance_matrix = inverse_matrix
covariance_matrix = np.array(covariance_matrix) # convert covariance matrix into numpy array

np.fill_diagonal(covariance_matrix, 0) # make sure there are no connections to the node's self

# Convert to NetworkX graphs
graph = nx.from_numpy_array(covariance_matrix)

# Iterate over the graph
for i, j in graph.edges():
    graph[i][j]['weight'] = round(covariance_matrix[i, j]) # round the weights to be a whole number for analysi

clustering_coefficient = nx.clustering(graph, weight='weight') # calculating the cluster coef
print("Clustering Coefficients:", clustering_coefficient) # display the cluster coef
# Get the minimum and maximum coefficients
min_coef = min(clustering_coefficient.values())
max_coef = max(clustering_coefficient.values())
print("Min coef", min_coef)
print("Max coef", max_coef)

widths = nx.get_edge_attributes(graph, 'weight') # get graph edge attributes
pos = nx.shell_layout(graph) # get the positions of the graph
nx.draw(graph,pos,with_labels=True) # draw the graph nodes
nx.draw_networkx_edges(graph, pos,edgelist=widths.keys(),width=list(widths.values())) # draw the graph edges

# Plot the degree distribution in the graph
degree_sequence = sorted((d for n, d in graph.degree(weight='weight')), reverse=True) # get weighted degree dis
dmin = min(degree_sequence) # get the min weighted degree value
dmax = max(degree_sequence) # get the max weighted degree value
fig = plt.figure(figsize=(8, 8)) # make the plot easily visible
axgrid = fig.add_gridspec(5, 4) # add axis grid
ax1 = fig.add_subplot(axgrid[3:, :2]) # make the first plot
ax1.plot(degree_sequence, "b-", marker="o") # add normal plot for the degree weights
ax1.set_title("Degree Rank Plot") # set the title
ax1.set_ylabel("Degree") # set the y-axis
ax1.set_xlabel("Rank") # set the x-axis
ax2 = fig.add_subplot(axgrid[3:, 2:]) # make the second plot
ax2.bar(*np.unique(degree_sequence, return_counts=True)) # add bar plot the the degree weights
ax2.set_title("Degree histogram") # set the title
ax2.set_xlabel("Degree") # set the y-axis
ax2.set_ylabel("# of Nodes") # set the x-axis
fig.tight_layout() # set the layout for easier visuals
plt.show() # display plot
```

```
Clustering Coefficients: {0: 0, 1: 0, 2: -0.01282051282051282, 3: -0.06666666666666667, 4: 0, 5: 0, 6: 0, 7: -0.01098901098901099, 8: 0, 9: 0, 10: 0, 11: 0, 12: 0, 13: 0, 14: 0, 15: 0, 16: 0, 17: 0, 18: 0, 19: 0, 20: -0.015151515151515152, 21: 0, 22: 0, 23: 0, 24: 0, 25: 0, 26: 0, 27: -0.03571428571428571, 28: 0, 29: 0, 30: 0, 31: 0, 32: -0.03571428571428571, 33: 0, 34: 0, 35: 0, 36: 0, 37: 0, 38: 0}
Min coef -0.06666666666666667
Max coef 0
```





Local Graph Measures EC Analysis

(1) The Clustering Coefficients are ranging from -0.0666 to 0, where 0 is for clusters with low corralation to one another and -0.06666 is for the higher weighted clusters with high (negative) corralation to one another. This is different from Q3 due to the minmum there being 0 and the maximum here being 0, showing us that the clustering coefficients here are dealing with more negative values. Compared to the EO case the minimum here is around 2x as large (both Q4 sparce inverse matrix) and that looks to be comming from the -3 degree value being set (not present in the EO case).

(2) Each of the nodes are verying in terms of connectivity some are well connected with 10+ edges, other are not with ~5 edges.Previously all nodes in Q3 seemed to all be connected to one another directly due to the density of the lines in the graph image.

(3.a) The Degree Rank Plot shows the the degree weight along with the rank of that node. Since there are 39 nodes there should roughly be 39 ranks and we can see that from the plot. Values of the weighted degrees are ranging from -3 to 2 which is 5 points in separation from the highest to lowest weith. This is 23 points less in comparision to EC Q3 (total 20 points) & meaning that the corralation matrix had

~5x more range in terms of corralation strength vs the inverse sparce matrix. However the range here is the same as the Q4 EO case, it's just shifted the range down by one.

(3.b) The Degree historgram shows the degree weight along with the number of nodes which is similar to the degree rank plot, but shows us a better picuture on the frequncy of each weighted degree. Showing us that degree 0 had the highest frequency rate at 22, telling us that there are a lot of no corralated nodes in the graph and this number is way higher that Q3 EC which was 5 at the highest and even higher than Q4 EO which was 18 at degree 0.

Global Graph Measures EC (eyes closed)

In [16]:

```
global_efficiency = nx.global_efficiency(graph)
average_clustering = nx.average_clustering(graph, weight='weight')

print(global_efficiency)
print(average_clustering)
print(nx.diameter(graph))
```

```
0.6079622132253729
-0.0045399045399045395
3
```

Global Graph Measures EC Analysis

(1) The Global efficiency of the graph is 0.607 which tells us that this is a relativly well connected graph considering the scale from 0 to 1, however this number is less than the perfectly connected graph we had in the corralation matrix case & just a little bit worse that what we had in the Q4 EO case (0.626).

(2) The average cluster of the graph is -0.00453 which tells what the average correlation of a cluster roughly is. This is a way smaller number than we had previously in Q3 (the lowest of both values being 0.03348) and that means there is way less corraltion going on in the inverse sparce matrix in comparision to the corralation matrix. However even then, this case still has 2x more corralation compared to Q4 EO, so the connections here are more extreme then that case.

(3) The diameter of the graph is 3 meaning the longest shortest path between any two nodes is 3. This is longer than the 1 we had in Q3 and this makes since because we don't have a perfectly connected graph like in the corralation matrix case.

Inverse Sparce Matrix vs Corralation Matrix

It seems that the inverse sparce matrix misses out on some relationships that the corralation matrix doesn't, however it shows the most important relation ships in a way that makes it easier for the reader to see (3 clear blue/red lines for example) vs the many dark red and blue lines you see in the corralation matrix. So in order to get the best understanding for what's happening, using both would be greatly benefitial because both of them have clear tradeoffs.