Consider I have a random matrix. Find the distribution of the eigenvalues given the distribution of elements in the matrix.

In [99]:

```
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
```

I can estimate the distribution of the eigenvalues by simulating the random variables and calculating the distribution of the eigenvalues by monte carlo method

In [2]:

```
samples = 100000
x1_array = np.random.normal(0, 1, samples)
x2_array = np.random.normal(0, 1, samples)
x3_array = np.random.normal(0, 0.1, samples)
x4_array = np.random.normal(0, 1, samples)
```

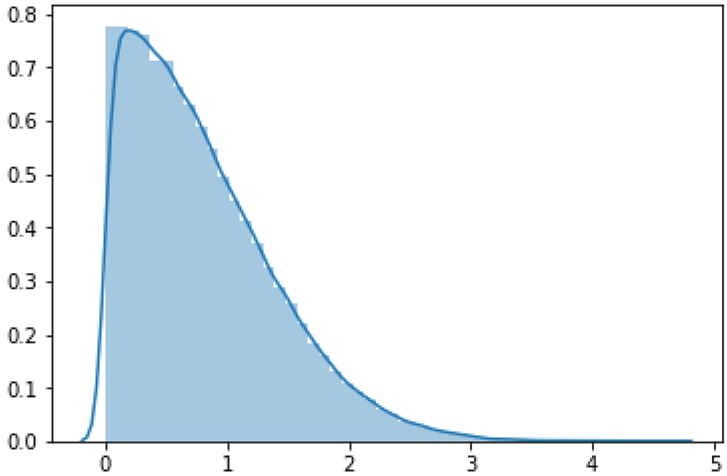
In [78]:

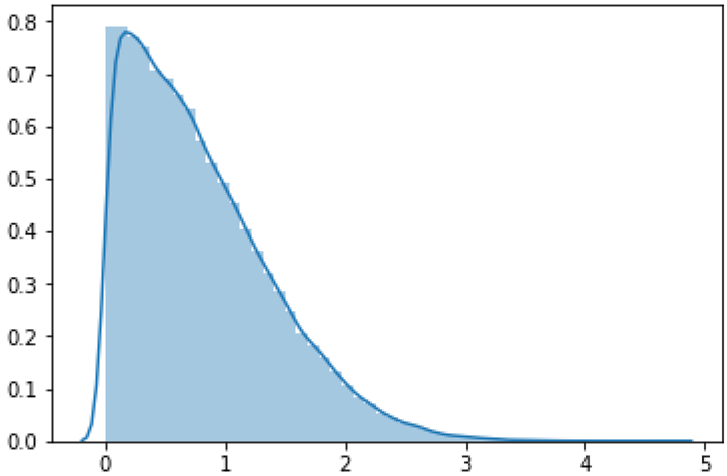
```
eigenvalues = np.zeros((samples,2),dtype=complex)

for i in range(0, samples):
    random_matrix = np.array([[x1_array[i], x3_array[i]], [x3_array[i], x4_array[i]]
    eigen = np.linalg.eigvals(random_matrix)
    eigen = np.sort_complex(eigen)
    eigenvalues[i] = eigen
print(eigenvalues.shape)
print(eigenvalues)
```

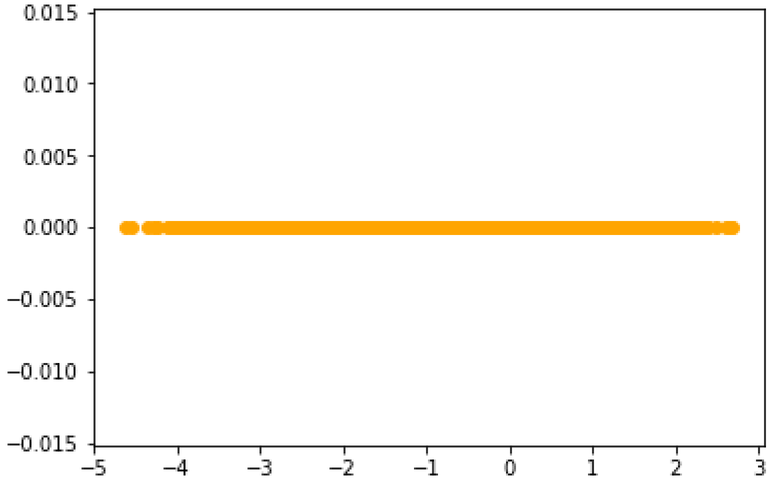
```
(100000, 2)
[[-0.17696448+0.j  1.10511139+0.j]
 [ 0.21461377+0.j  2.41447478+0.j]
 [-2.72222464+0.j -0.39274523+0.j]
 ...
 [ 0.3755242 +0.j  1.76707986+0.j]
 [ 0.49667753+0.j  0.93701772+0.j]
 [-1.4245597 +0.j -0.75924597+0.j]]
```

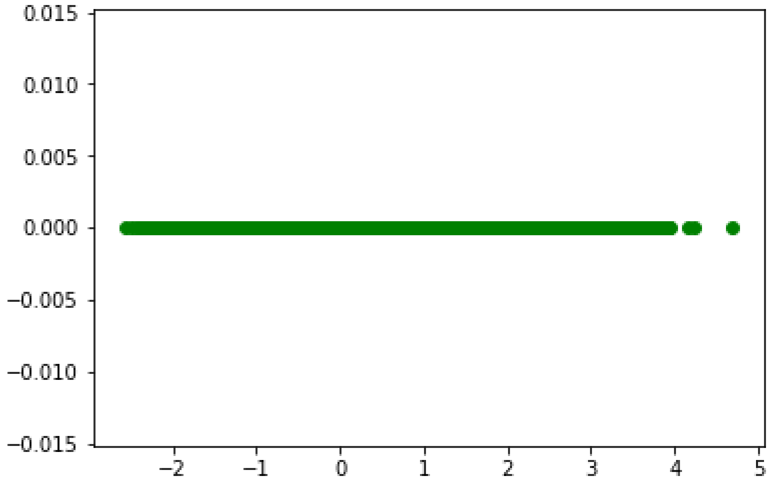
In [79]:

```
X = eigenvalues.real
Y = eigenvalues.imag
Z = np.power(np.power(X,2) + np.power(Y,2), 0.5)
sns.distplot(Z[:, 0])
plt.show()
sns.distplot(Z[:, 1])
plt.show()
plt.scatter(X[:, 0],Y[:, 0], color = 'orange')

plt.show()
plt.scatter(X[:, 1],Y[:, 1], color = 'green')
plt.show()
```









Distribution of eigenvalues in $N \times N$ matrices

In [142]:

```
# Simulate an n * n iid random matrix with gaussian random variable
def mesh(reduced_data, kmeans, count):
    h = .02     # point in the mesh [x_min, x_max]x[y_min, y_max].

    # Plot the decision boundary. For that, we will assign a color to each
    x_min, x_max = reduced_data[:, 0].min() - 1, reduced_data[:, 0].max() + 1
    y_min, y_max = reduced_data[:, 1].min() - 1, reduced_data[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    # Obtain labels for each point in mesh. Use last trained model.
    Z = kmeans.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.figure(1)
    plt.clf()
    plt.imshow(Z, interpolation='nearest',
               extent=(xx.min(), xx.max(), yy.min(), yy.max()),
               cmap=plt.cm.Paired,
               aspect='auto', origin='lower')

    plt.plot(reduced_data[:, 0], reduced_data[:, 1], 'k.', markersize=2)
    # Plot the centroids as a white X
    centroids = kmeans.cluster_centers_
    plt.scatter(centroids[:, 0], centroids[:, 1],
                marker='x', s=169, linewidths=3,
                color='w', zorder=10)
    plt.title('K-means clustering on the digits dataset (PCA-reduced data)\n'
              'Centroids are marked with white cross')
    plt.xlim(x_min, x_max)
    plt.ylim(y_min, y_max)
    plt.xticks(())
    plt.yticks(())
    filename = "figure" + str(count) + ".png"
#     plt.savefig(filename, bbox_inches='tight')
    plt.show()
```

In [160]:

```
n = int(input("Tell me the shape of the matrix: "))
from sklearn.svm import SVC as SVM
sample = 100000
def plot_svc_decision_function(model, ax=None, plot_support=True):
    """Plot the decision function for a 2D SVC"""
    if ax is None:
        ax = plt.gca()
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()

    # create grid to evaluate model
    x = np.linspace(xlim[0], xlim[1], 30)
    y = np.linspace(ylim[0], ylim[1], 30)
    Y, X = np.meshgrid(y, x)
    xy = np.vstack([X.ravel(), Y.ravel()]).T
    P = model.decision_function(xy).reshape(X.shape)

    # plot decision boundary and margins
    ax.contour(X, Y, P, colors='k',
               levels=[-1, 0, 1], alpha=0.5,
               linestyles=['--', '-', '--'])

    # plot support vectors
    if plot_support:
        ax.scatter(model.support_vectors_[:, 0],
                   model.support_vectors_[:, 1],
                   s=300, linewidth=1, facecolors='none');
    ax.set_xlim(xlim)
    ax.set_ylim(ylim)
    plt.show()

coefficient_matrix = np.array([[0, 0]])
for j in range(sample):
    matrix = np.random.normal(0, 1, n*n)
    matrix = np.reshape(matrix, (n, n))

    # Applying Singular Value Decomposition on the above matrix
    U, S, V_t = np.linalg.svd(matrix, full_matrices=False)
    S = np.diag(S)
    eigenvectors = np.dot(matrix, V_t.T)
#     print(eigenvectors)
    # Select the first two principal components
    two_principal = eigenvectors[:, :2]

    X = two_principal[:, 0]
    Y = two_principal[:, 1]
    X_new = np.array([[X[0], Y[0]]])

    for i in range(1, X.shape[0]):
        X_new = np.vstack((X_new, np.array([X[i], Y[i]])))

    kmeans = KMeans(n_clusters=2, random_state=0).fit(X_new)
    centroids = kmeans.cluster_centers_
    labels = kmeans.labels_
    SVC_class = SVM(kernel='linear')
    SVC_class.fit(X_new, labels)
#     print(type(SVC_class.coef_))
#     print(SVC_class.intercept_)
#     print("The Boundaries are shown below")
```

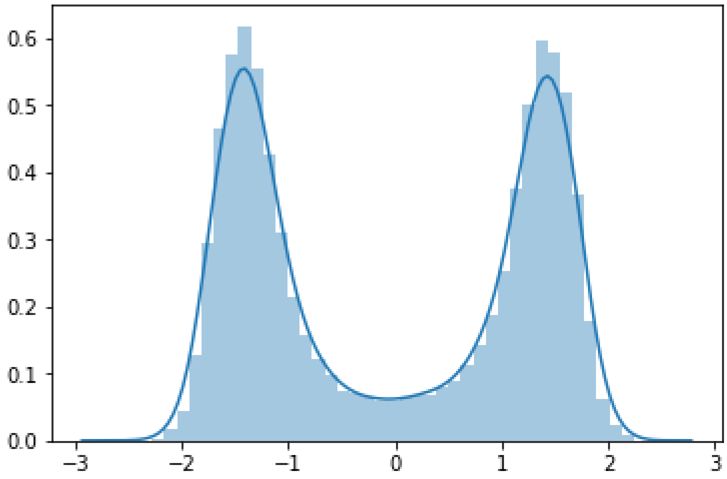
```
#     mesh(two_principal, kmeans, j)
#     plt.scatter(X_new[:, 0], X_new[:, 1], c=labels, s=50, cmap='autumn')
#     plot_svc_decision_function(SVC_class)
    print("Sampling: ", j)
    coefficient_matrix = np.vstack((coefficient_matrix, SVC_class.coef_))

coefficient_matrix = coefficient_matrix[1:, :]
print(coefficient_matrix)
```

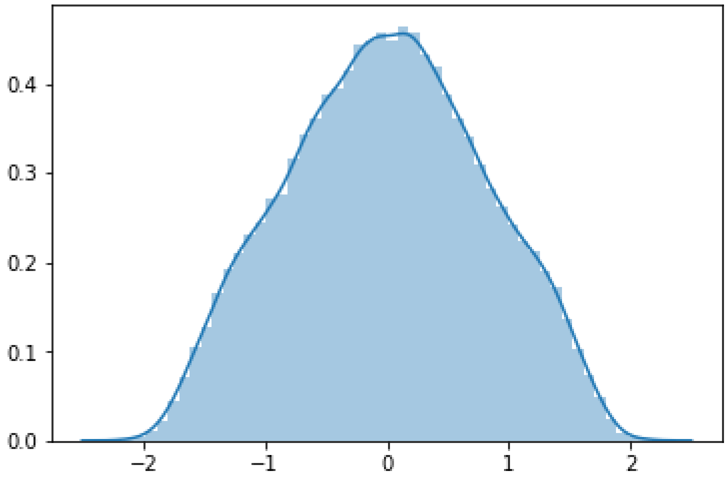
```
Tell me the shape of the matrix: 25
Sampling:  0
Sampling:  1
Sampling:  2
Sampling:  3
Sampling:  4
Sampling:  5
Sampling:  6
Sampling:  7
Sampling:  8
Sampling:  9
Sampling:  10
Sampling:  11
Sampling:  12
Sampling:  13
Sampling:  14
Sampling:  15
Sampling:  16
Sampling:  17
```

In [161]:

```
sns.distplot(coefficient_matrix[:, 0])
plt.show()
sns.distplot(coefficient_matrix[:, 1])
plt.show()
```

```
/Users/apple/anaconda3/envs/env-python3/lib/python3.6/site-packages/sc
ipy/stats/stats.py:1713: FutureWarning: Using a non-tuple sequence for
multidimensional indexing is deprecated; use `arr[tuple(seq)]` instead
of `arr[seq]`. In the future this will be interpreted as an array inde
x, `arr[np.array(seq)]`, which will result either in an error or a dif
ferent result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```





In [ ]:

```

```