Solution1: Assuming that the result holds for M dimensionality. The M+1 Dimensional subspace is governed by M principle eigen vectors and additional M+1 eigen vector that needs to be orthogonal to the previous $(u_1, u_2 ... u_m)$. We apply lagrangian multiplier to contrain this. We know that by variance formula, that variance for M+1 $= u_{M+1}^T S u_{M+1}$ Applying lagrangian multiplier, we get the following function to maximise:
$u_{M+1}^T S u_{M+1} + \lambda_{M+1}(1 - u_{M+1}^T u_{M+1}) \Sigma \eta u_i u_{M+1}$
Taking a derivative of above function wrt u_{M+1} we get:
$2Su_{M+1} - 2\lambda_{M+1} u_{M+1} + \Sigma_{i=1}^{M} \eta_i u_i$
Simplifying this by multiplying by u_j^T on both sides, we see that η=0 for j between 1,..M. Therefore, we obtain:
$Su_{M+1} = \lambda_{M+1} u_{M+1}$
Clearly, u_{M+1} is an eigen vector of S with eigen value λ_{M+1}. Thus the variance is maximised by selecting eigen vector u_{M+1} with largest eigen value λ_{M+1}.

Solution2: The log-likelihood function is given by:
$L(\mu, W, \Phi) =$
$\frac{-ND}{2} ln(2\pi) - \frac{N}{2} ln|WW^T + \Phi| - \frac{1}{2}\Sigma_{n=1}^{N}(x_n - \mu)^T (WW^T + \Phi)^{-1}(x_n - \mu)$
Now the log likelihood function for the transformed data can be given as:
$L_A(\mu, W, \Phi) =$
$-\frac{ND}{2} ln(2\pi) - \frac{N}{2} ln|WW^T + \Phi| - \frac{1}{2}\Sigma_{n=1}^{N}(Ax_n - \mu)^T (WW^T + \Phi)^{-1}(Ax_n - \mu)$
Solving for μ:
$\mu_A = \frac{1}{N}\Sigma_{n=1}^{N} Ax_n = Ax' = A\mu$
Substituting this back in the log-likelihood function and also the values for Φ_A and W_A,we get:
$L_A(\mu_A, W_A, \Phi_A) = -\frac{ND}{2} ln(2\pi) - \frac{N}{2} ln|W_A W_A^T + \Phi_A| -$
$\frac{1}{2}\Sigma_{n=1}^{N}(x_n - \mu_A)^T (W_A W_A^T + \Phi_A)^{-1}(x_n - \mu_A) - Nln|A|$
This looks exactly like the log-likelihood function except for the last term.

Solution3: We can map Leslie's choices [0,3,0,0,4] into concept space by multiplying it by the 5x5 V matrix, getting the form [1.74,2.84]. Multiplying this by V^T, we get [1.0092,1.0092,1.0092,2.0164,2.0164] which is useful to gauge how Leslie likes the other movies.

Programming assignment 10: Dimensionality Reduction

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt

%matplotlib inline
```

PCA Task

Given the data in the matrix X your tasks is to:

- Calculate the covariance matrix Σ.
- Calculate eigenvalues and eigenvectors of Σ.
- Plot the original data X and the eigenvectors to a single diagram. What do you observe? Which eigenvector corresponds to the smallest eigenvalue?
- Determine the smallest eigenvalue and remove its corresponding eigenvector. The remaining eigenvector is the basis of a new subspace.
- Transform all vectors in X in this new subspace by expressing all vectors in X in this new basis.

The given data X

In [2]:
```
X = np.array([(-3,-2),(-2,-1),(-1,0),(0,1),
              (1,2),(2,3),(-2,-2),(-1,-1),
              (0,0),(1,1),(2,2), (-2,-3),
              (-1,-2),(0,-1),(1,0), (2,1),(3,2)])
```

Task 1: Calculate the covariance matrix Σ

In [73]:
```
def get_covariance(X):
    """Calculates the covariance matrix of the input data.

    Parameters
    ----------
    X : array, shape [N, D]
        Data matrix.

    Returns
    -------
    Sigma : array, shape [D, D]
        Covariance matrix

    """
    Sigma = np.cov(np.transpose(X))

    return Sigma
```

Task 2: Calculate eigenvalues and eigenvectors of Σ.

In [74]:
```
def get_eigen(S):
    """Calculates the eigenvalues and eigenvectors of the input matrix.

    Parameters
    ----------
    S : array, shape [D, D]
        Square symmetric positive definite matrix.

    Returns
    -------
    L : array, shape [D]
        Eigenvalues of S
    U : array, shape [D, D]
        Eigenvectors of S

    """
    L,U = np.linalg.eig(S)
    return L,U
```

Task 3: Plot the original data X and the eigenvectors to a single diagram.

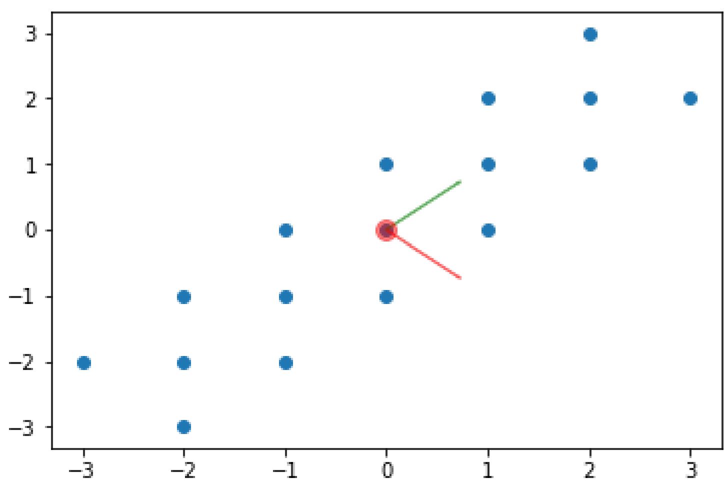
In [75]:
```
# plot the original data
plt.scatter(X[:, 0], X[:, 1])

# plot the mean of the data
mean_d1, mean_d2 = X.mean(0)
plt.plot(mean_d1, mean_d2, 'o', markersize=10, color='red', alpha=0.5)

# calculate the covariance matrix
Sigma = get_covariance(X)
#print("Sigma is", Sigma)
# calculate the eigenvector and eigenvalues of Sigma
L, U = get_eigen(Sigma)
#print("L : ",L)
#print("U is: ", U)

plt.arrow(mean_d1, mean_d2, U[0, 0], U[0, 1], width=0.01, color='red', alpha=
0.5)
plt.arrow(mean_d1, mean_d2, U[1, 0], U[1, 1], width=0.01, color='green', alpha
=0.5);
```

```
L :  [ 5.625  0.375]
U is:  [[ 0.70710678 -0.70710678]
 [ 0.70710678  0.70710678]]
```



What do you observe in the above plot? Which eigenvector corresponds to the smallest eigenvalue?

Write your answer here:

The eigenvector U[1] corresponds to the smaller eignevalue, 0.375

Task 4: Transform the data

Determine the smallest eigenvalue and remove its corresponding eigenvector. The remaining eigenvector is the basis of a new subspace. Transform all vectors in X in this new subspace by expressing all vectors in X in this new basis.

In [89]:
```
def transform(X, U, L):
    """Transforms the data in the new subspace spanned by the eigenvector corr
esponding to the largest eigenvalue.

    Parameters
    ----------
    X : array, shape [N, D]
        Data matrix.
    L : array, shape [D]
        Eigenvalues of Sigma_X
    U : array, shape [D, D]
        Eigenvectors of Sigma_X

    Returns
    -------
    X_t : array, shape [N, 1]
        Transformed data

    """
    L_list = L.tolist()
    U_list = U.tolist()
    position = L_list.index(min(L))
    U_list.pop(position)
    U_fin = np.asarray(U_list)
    #print(U_fin.shape)

    X_t = np.dot(X,np.transpose(U_fin))

    return X_t
```

In [90]:
```
X_t = transform(X, U, L)
#print(X_t)
```

Task SVD

Task 5: Given the matrix M find its SVD decomposition $M = U \cdot \Sigma \cdot V$ and reduce it to one dimension using the approach described in the lecture.

In [92]:
```
M = np.array([[1, 2], [6, 3],[0, 2]])
```

In [112]:

```
def reduce_to_one_dimension(M):
    """Reduces the input matrix to one dimension using its SVD decomposition.

    Parameters
    ----------
    M : array, shape [N, D]
        Input matrix.

    Returns
    -------
    M_t: array, shape [N, 1]
        Reduce matrix.

    """
    U,S,V = np.linalg.svd(M,full_matrices=False)
    Sigma = np.diag(S)

    M_t = np.dot(U,Sigma)
    return M_t
```

In [113]:

```
M_t = reduce_to_one_dimension(M)
print(M_t)
```

```
[[-1.90211303  1.1755705 ]
 [-6.68109819 -0.60243425]
 [-1.05146222  1.70130162]]
```