Kalman Filter

First, I initialized the State matrix with values he provided. I also initialized P as the Estimation Covariance matrix, with error terms that correspond to the variance of the x position and x velocity, specific to the estimates. Then, for each observation that was provided, I iterate through a series of processes to update the state matrix with values provided by the Kalman filter. First, I make a prediction of where the plane will be in the next time step. This prediction is simply based on the previous position and velocity, with an acceleration parameter that adjusts the velocity. The A matrix in the prediction2d function updates the previous state based on the time that has elapsed. The B matrix translates the acceleration into an adjustment to the position and velocity. The formula 1/2 time squared is used to find a distance given an acceleration. Then, I updated the Process / Estimation Covariance matrix to the next time step, predicting it forward. The operation adds a time step to the matrix, subsequently updating the variance of the distance error. The A matrix is similar to the one used in predicting the State matrix values. Since Professor Biezen eliminated the off-diagonal values, I did the same. Calculating the Kalman gain involved calculating the covariance matrix for the observation errors, and using it to compare with the process covariance matrix. In the problem, there's probably more matrix rotation than what's required, but I believe it's meant to make the formula invariant to different matrix sizes. There is no division in matrix operations, so to find the ratio I used the dot product with the inverse of what would otherwise be the denominator. Notice that in matrix format, the Kalman gain is a matrix of the same dimension as the inputs, and along the diagonal are weights that adjust the observed position and velocity. The lower the weights, the lower the model trusts the observations compared to the predictions. With the newly calculated Kalman gain, I used it to weigh the difference between the observation data with the prediction, which was then used to update the state matrix. I also used the Kalman gain to update the process covariance matrix. I did that as many times as observations that were provided, and got a result that is somewhere between the observation data and predictive model, which is exactly what one should expect from using the Kalman filter. My exact results were slightly different than Professor Biezen, but note that he did commit a number of errors in his calculations and did not do a full run through of all the observations, so I'm confident my calculations were more accurate. Source: James Teow

```
import numpy as np
from numpy.linalg import inv

x_observations = np.array([4000, 4260, 4550, 4860, 5110])
v_observations = np.array([280, 282, 285, 286, 290])

z = np.c_[x_observations, v_observations]

# Initial Conditions
a = 2  # Acceleration
v = 280
t = 1  # Difference in time

# Process / Estimation Errors
error_est_x = 20
error_est_v = 5

# Observation Errors
error_obs_x = 25  # Uncertainty in the measurement
error_obs_v = 6

def prediction2d(x, v, t, a):
    A = np.array([[1, t],
                  [0, 1]])
    X = np.array([[x],
                  [v]])
    B = np.array([[0.5 * t ** 2],
                  [t]])
    X_prime = A.dot(X) + B.dot(a)
    return X_prime


def covariance2d(sigma1, sigma2):
    cov1_2 = sigma1 * sigma2
    cov2_1 = sigma2 * sigma1
    cov_matrix = np.array([[sigma1 ** 2, cov1_2],
                           [cov2_1, sigma2 ** 2]])
    return np.diag(np.diag(cov_matrix))


# Initial Estimation Covariance Matrix
P = covariance2d(error_est_x, error_est_v)
A = np.array([[1, t],
              [0, 1]])

# Initial State Matrix
X = np.array([[z[0][0]],
              [v]])
n = len(z[0])

for data in z[1:]:
    X = prediction2d(X[0][0], X[1][0], t, a)
    # To simplify the problem, professor
    # set off-diagonal terms to 0.
    P = np.diag(np.diag(A.dot(P).dot(A.T)))

    # Calculating the Kalman Gain
    H = np.identity(n)
    R = covariance2d(error_obs_x, error_obs_v)
    S = H.dot(P).dot(H.T) + R
    K = P.dot(H).dot(inv(S))

    # Reshape the new data into the measurement space.
    Y = H.dot(data).reshape(n, -1)

    # Update the State Matrix
    # Combination of the predicted state, measured values, covariance matrix and Kalman Gain
    X = X + K.dot(Y - H.dot(X))

    # Update Process Covariance Matrix
    P = (np.identity(len(K)) - K.dot(H)).dot(P)

print("Kalman Filter State Matrix:\n", X)
```

```
Kalman Filter State Matrix:
 [[5127.05898493]
 [ 288.55147059]]
```