Assignment 1 (part I): Line Fitting

Problem 1

Prove that affine transformations preserve parallel lines.

Solution: Suppose we have two parallel lines $\vec{a} = (Q - R)$ and $\vec{b} = b(Q - R)$ where Q and R are points in the affine space, and b is some scalar. Consider applying some affine transformation T. Since affine transformations must be linear transformations on vectors ($(L(a\vec{b}) = aL(\vec{b})$) applying T to $\vec{a}$ and $\vec{b}$ yields:

$$T(\vec{a}) = T(Q) - T(R)$$

, applying the transform to each of the points of $\vec{a}$

$$T(\vec{b}) = T(b\vec{a}) = bT(\vec{a})$$

Hence, $T(\vec{a})$ and $T(\vec{b})$ are also parallel. QED.

Problem 2 (least-squares)

Complete implementation of function $estimate$ of class $LeastSquareLine$ below. It should update line parameters a and b correpsonding to line model $y = ax + b$. You can use either SVD of matrix A or inverse of matrix $A^T A$, as discussed in class. NOTE: several cells below test your code.

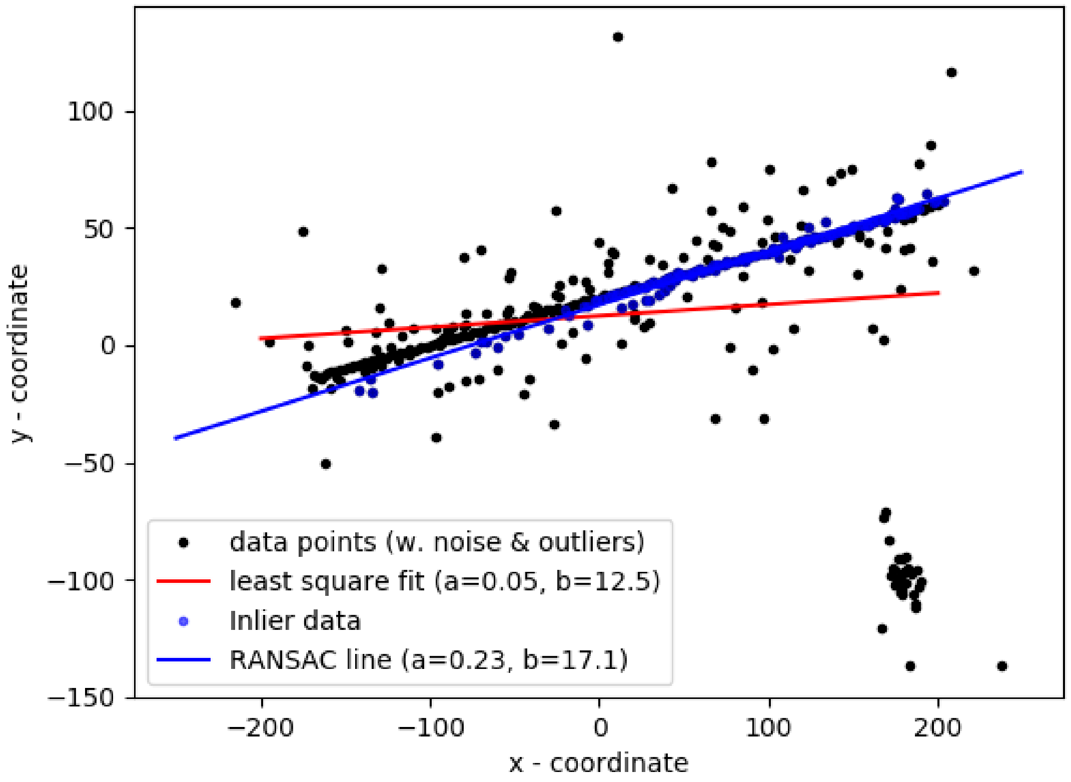
In [1]:
```
%matplotlib notebook

import numpy as np
import numpy.linalg as la
import matplotlib
import matplotlib.pyplot as plt
from skimage.measure import ransac
```

In [2]:
```
class LeastSquareLine:

    def __init__(self):
        self.a = 0.0
        self.b = 0.0

    def estimate(self, points2D):
        B = points2D[:,1]
        A = np.copy(points2D)
        A[:,1] = 1.0
        # Vector B and matrix A are already defined. Change code below
        self.a = 0.0
        self.b = 0.0
        psInv = la.pinv(A)
        #'matrix'...difference between dot and multiplying is annoying to figu
re out :p
        product = psInv.dot(B);
        self.a = product[0]
        self.b = product[1]
        return True

    def predict(self, x): return (self.a * x) + self.b

    def predict_y(self, x): return (self.a * x) + self.b

    def residuals(self, points2D):
        return points2D[:,1] - self.predict(points2D[:,0])

    def line_par(self):
        return self.a, self.b
```

Problem 3 (RANSAC for robust line fitting, single model)

Working code below generates a noisy cloud of points in $\mathcal{R}^2$ from a given line and a group of outliers.

In [3]:
```
np.random.seed(seed=1)

# parameters for "true" line y = a*x + b
a, b = 0.2, 20.0

# x-range of points [x1,x2]
x_start, x_end = -200.0, 200.0

# generate the "true" line points
x = np.arange(x_start,x_end)
y = a * x + b
data = np.column_stack([x, y])    # staking data points into (Nx2) array
# add faulty data (i.e. outliers)
faulty = np.array(30 * [(180., -100)])  # (30x2) array containing 30 rows [18
0,-100]  (points)
faulty += 5 * np.random.normal(size=faulty.shape)  # adding Gaussian noise to
 these points
data[:faulty.shape[0]] = faulty   # replacing the first 30 points in data with
 faulty (outliers)

# add gaussian noise to coordinates
noise = np.random.normal(size=data.shape) # generating Gaussian noise (varianc
e 1) for each data point (rows in 'data')
data += 0.5 * noise
data[::2] += 5 * noise[::2]  # every second point adds noise with variance 5
data[::4] += 20 * noise[::4] # every fourth point adds noise with variance 20

fig, ax = plt.subplots()
ax.plot(data[:,0], data[:,1], '.k', label='data points (w. noise & outliers)')
ax.set_xlabel('x - coordinate')
ax.set_ylabel('y - coordinate')
ax.legend(loc='lower left')
plt.show()
```



Code below uses your implementation of class $LeastSquareLine$ for least-square line fitting for the data above. The estimated line is displayed in the cell above. Use this cell to test your code in Problem 2. Of course, your least-square line will be affected by the outliers.

In [4]:
```
LSline = LeastSquareLine() # uses class implemented in Problem 2
print LSline.estimate(data)
a_ls, b_ls = LSline.line_par()

# visualizing estimated line
ends = np.array([x_start,x_end])
ax.plot(ends, LSline.predict(ends), '-r', label='least square fit (a={:4.2f},
 b={:4.1f})'.format(a_ls,b_ls))
ax.legend(loc='lower left')
plt.show()
```

```
True
```

(part a) Assume that a set of $N = 100$ points in $2D$ includes $N_i = 20$ inliers for one line and $N_o = 80$ outliers. What is the least number of times one should sample a random pair of points from the set to get probability $p \geq 0.95$ that in at least one of the sampled pairs both points are inliers? Derive a general formula and compute a numerical answer for the specified numbers.

Solution: Given N_i inliers and N points, the probability of picking 2 outliers randomly on a given selection (assuming they are chosen sequentially) is $(N_i/N) * (N_i - 1)/(N - 1)$. Assuming each of our pair selections is independent (we don't remove previously selected pairs from the set of points), after x pair choices, the chance of finding at least 1 pair of inliers is:

$$P(N_i, N) = 1 - (1 - ((N_i/N) * (N_i - 1)/(N - 1)))^x$$

Note: Round up to the nearest integer to get the actual number of iterations 1 minus the probability that none are found for each of the x choices. Then for the specific example given:

$$0.95 = 1 - (1 - (20/100) * (19/99))^x$$

(Simplify and use log rules)

$$x = log(0.05)/log(0.9616) = 76.5 \implies 77$$

Hence, we should sample at least 77 times to get a >= 95% chance of both points being inliers

(part b) Using the knowledge of the number of inliers/outliers in the example at the beginning of Problem 3, estimate the minimum number of sampled pairs needed to get RANSAC to "succeed" (to get at least one pair of inliers) with $p \geq 0.95$. Use your formula in part (a). Show your numbers in the cell below. Then, use your estimate as a value of parameter max_trials inside function $ransac$ in the code cell below and test it. You should also change $residual\_threshold$ according to the noise level for inliers in the example. NOTE: the result is displayed in the same figure at the beginning of Problem 3.

Your estimates: Based on data, N = 400, N_i = 300 (considering every 4th point w/ noise above 5 to be an outlier, which will inlcude the 30 outliers)

$$0.95 = 1 - (1 - (300/400) * (299/399))^x x = log(0.05)/log(0.43) = 3.62 \implies 4$$

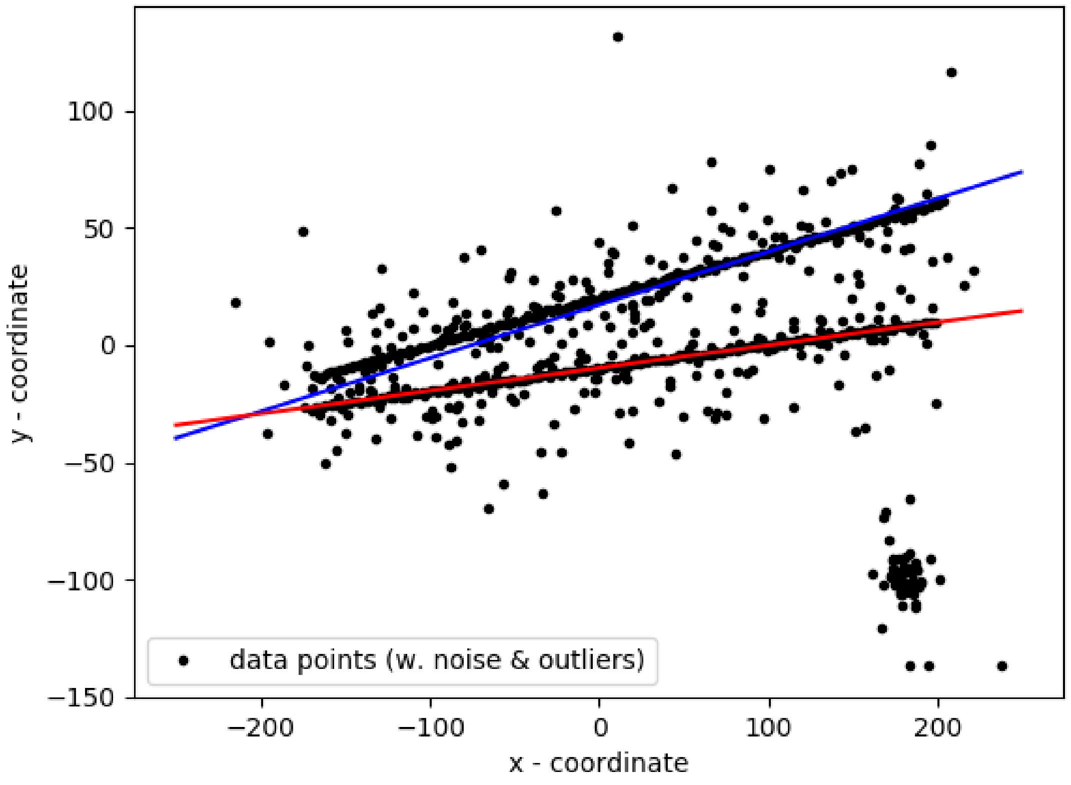
Given our outlier definition, we set 5 as our residual threshold to keep points w/ small noise as non-outliers.

In [5]:
```
# robustly fit line using RANSAC algorithm
model_robust, inliers = ransac(data, LeastSquareLine, min_samples=2, residual_
threshold=5,max_trials=4)
a_rs, b_rs = model_robust.line_par()

# generate coordinates of estimated models
line_x = np.arange(-250, 250)
line_y_robust = model_robust.predict_y(line_x)

#fig, ax = plt.subplots()
ax.plot(data[inliers, 0], data[inliers, 1], '.b', alpha=0.6, label='Inlier dat
a')
ax.plot(line_x, line_y_robust, '-b', label='RANSAC line (a={:4.2f}, b={:4.1
f})'.format(a_rs,b_rs))
ax.legend(loc='lower left')
plt.show()
```

Problem 4 (sequential RANSAC for robust multi-line fitting)

Generating noisy data with outliers

In [6]:
```
# parameters for "true" lines y = a*x + b
a2, b2 = 0.1, -10.0

# generate the "true" line points
y2 = a2 * x + b2
data2 = np.column_stack([x, y2])    # staking data points into (Nx2) array

# add faulty data (i.e. outliers)
faulty = np.array(30 * [(180., -100)])  # (30x2) array containing 30 rows [18
0,-100]  (points)
faulty += 5 * np.random.normal(size=faulty.shape)  # adding Gaussian noise to
 these points
data2[:faulty.shape[0]] = faulty   # replacing the first 30 points in data2 wi
th faulty (outliers)

# add gaussian noise to coordinates
noise = np.random.normal(size=data.shape) # generating Gaussian noise (varianc
e 1) for each data point (rows in 'data')
data2+= 0.5 * noise
data2[::2] += 5 * noise[::2]  # every second point adds noise with variance 5
data2[::4] += 20 * noise[::4] # every fourth point adds noise with variance 20

data = np.concatenate((data,data2)) # combining with previous data

fig, ax = plt.subplots()
ax.plot(data[:,0], data[:,1], '.k', label='data points (w. noise & outliers)')
ax.set_xlabel('x - coordinate')
ax.set_ylabel('y - coordinate')
ax.legend(loc='lower left')
plt.show()
```



Write code below using sequential RANSAC to detect two lines in the data above. Your lines should be displayed in the figure above.

In [7]:
```
#from skimage.measure import ransac
# robustly fit line using RANSAC algorithm
LSline = LeastSquareLine() # uses class implemented in Problem 2
model_robust, inliers = ransac(data, LeastSquareLine, min_samples=2, residual_
threshold=12.5, max_trials=10)
a_rs, b_rs = model_robust.line_par()
ax.plot(line_x, line_y_robust, '-b')
newData = []
for i in range(len(data)):
    #Remove inliers from data
    if(inliers[i]):
        np.delete(data,i);
#data = newData;
line_x = np.arange(-250, 250)
line_y_robust = model_robust.predict_y(line_x)
#Use RANSAC again to predict second line
model_robust2, inliers2 =  ransac(data, LeastSquareLine, min_samples=2, residu
al_threshold=12.5, max_trials=10)
line_y_robust2 = model_robust.predict_y(line_x);
ax.plot(line_x, line_y_robust2, '-r')
```

```
data: 800
```

Out[7]:
```
[<matplotlib.lines.Line2D at 0xa221668>]
```