第四章 非参数滤波

1. 对前面章节讨论的线性系统进行直方图滤波

a). 对3.1的动态系统进行直方图滤波，绘制$t = 1, 2, \ldots, 5$每个时刻的$x, \dot{x}$联合概率分布

根据3.1的分析，可以得到

$$A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

$$B = \begin{bmatrix} \frac{1}{2} \\ 1 \end{bmatrix}$$

$$R = \begin{bmatrix} \frac{1}{4} & \frac{1}{2} \\ \frac{1}{2} & 1 \end{bmatrix} + \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \times 10^{-5}$$

注：其中协方差矩阵 R 是奇异矩阵不可逆，为计算高斯分布的概率，需要在该矩阵上加小量使其可逆

In [1]:

```
import numpy as np
import math
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

A = np.mat([[1, 1],[0, 1]])
B = np.mat([[0.5], [1.0]])
mu = np.mat([[0],[0]])
R = B*B.T + np.ones(2)*1e-5
```

为进行直方图滤波，首先需要对状态空间进行离散化，这里将$\begin{bmatrix} x \\ \dot{x} \end{bmatrix}$划分在 $-10 \sim 10$ 的空间，步长取 0.25，时刻t的概率分布由下式给定

$$p(x_t) = \sum_{x_{min}}^{x_{max}} \sum_{\dot{x}_{min}}^{\dot{x}_{max}} p(x_t|x_{t-1}) * p(x_{t-1})$$

其中状态转移函数$p(x_t|x_{t-1})$与3.1中一致，系统初始概率分布由 $\Sigma_0 = \begin{bmatrix} 10^{-10} & 0 \\ 0 & 10^{-10} \end{bmatrix}$ 确定

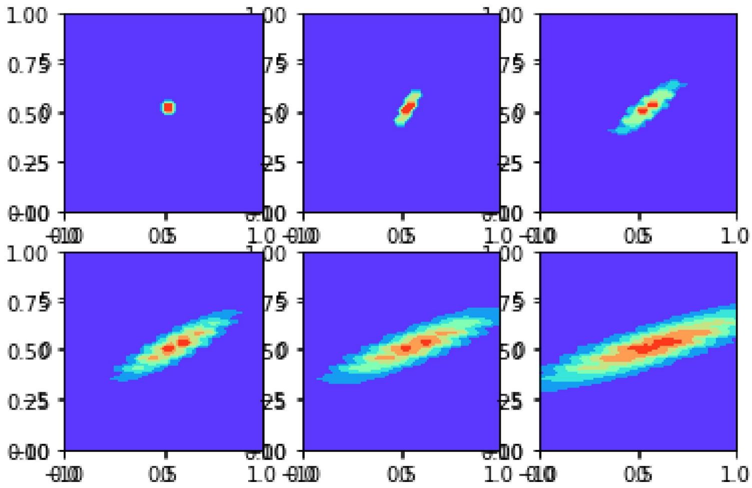
Loading [MathJax]/jax/output/HTML-CSS/jax.js

In [2]:

```
Sigma_0 = np.mat([[1e-5, 0],[0, 1e-5]])

delta = 0.5
X_max = 10
X_dot_max = 10
X_min = -10
X_dot_min = -10

X=np.arange(X_min, X_max, delta)
X_dot=np.arange(X_dot_min, X_dot_max, delta)
X, X_dot=np.meshgrid(X, X_dot)
P=np.zeros(X.shape)
#initialize P_0
i=math.floor(X.shape[0]/2)
j=math.floor(X.shape[1]/2)
P[i, j] = 0.25
P[i+1, j] = 0.25
P[i, j+1] = 0.25
P[i+1, j+1] = 0.25

fig,ax = plt.subplots(2,3)

#from time 0 to 5
for fi in range(6):
    print("figure", fi)
    ax = fig.add_subplot(2,3,1+fi)
    P_bar=np.zeros(X.shape)

    # calculate probabilitics
    sum = 0.0
    for i in range(P_bar.shape[0]):
        for j in range(P_bar.shape[1]):
            X_t=np.mat([[X[i,j]] , [X_dot[i,j]]])
            for i_t_1 in range(P.shape[0]):
                for j_t_1 in range(P.shape[1]):
                    X_t_1=np.mat([[X[i_t_1,j_t_1]], [X_dot[i_t_1,j_t_1]]])
                    P_bar[i][j]+=math.pow(math.e, -1.0/2.0 * (X_t-A*X_t_1).T * R.I * (X_t-A*X_t_
1))*P[i_t_1, j_t_1]
            sum+=P_bar[i][j]
    P_bar=P_bar/sum

    ax.contourf(X, X_dot, P, cmap='rainbow')
    P = P_bar

plt.show()
```

Loading [MathJax]/jax/output/HTML-CSS/jax.js

```
figure 0
figure 1
figure 2
figure 3
figure 4
figure 5
```



b). 对3.2的观测步骤进行直方图滤波，对比观测更新前和更新后的概率分布

根据3.2 可以得到观测矩阵以及观测概率函数

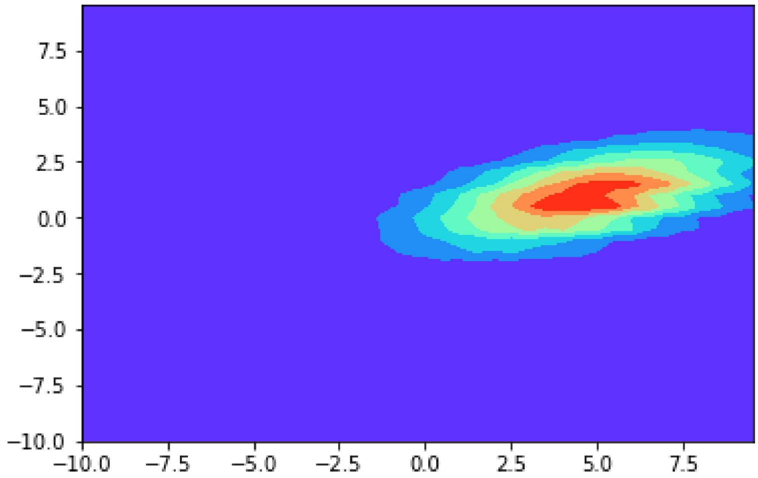
$$C = \begin{bmatrix} 1 & 0 \end{bmatrix}$$
$$Q = 10$$
$$p(z_t | x_t) \sim N(Cx_t, Q)$$

通过测量跟新可以得到新的概率分布如下

Loading [MathJax]/jax/output/HTML-CSS/jax.js

In [3]:

```
C = np.mat([1, 0])
Q = np.mat([10])
Z = 5
P_z = np.zeros(P.shape)
sum = 0.0
for i in range(P_z.shape[0]):
    for j in range(P_z.shape[1]):
        X_t=np.mat([[X[i][j]], [X_dot[i][j]]])
        P_z[i, j] = math.pow(math.e, -1.0/2.0 * (Z-C*X_t).T * Q.I * (Z-C*X_t))*P[i, j]
        sum+=P_z[i, j]
P_z=P_z/sum

fig=plt.figure()
ax=fig.add_subplot(111)
ax.contourf(X, X_dot, P_z, cmap='rainbow')
plt.show()
```



2. 对习题3.4进行直方图滤波实现

a). 为直方图滤波建议一个合适的初值估计

初值估计由三维高斯分布给定，分布的均值和方差如下

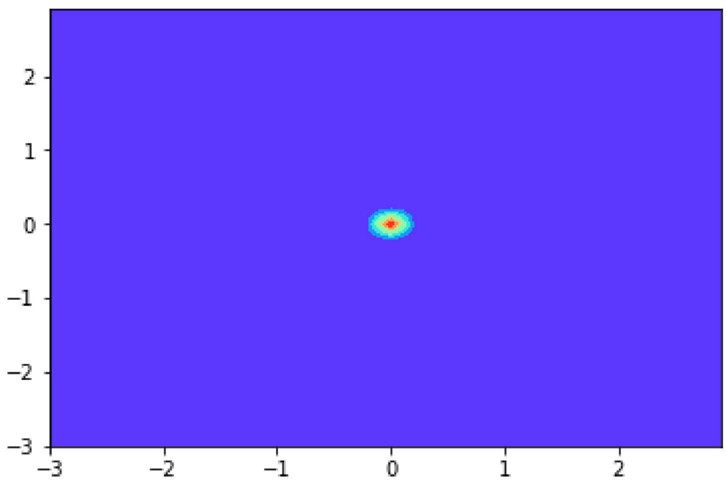
$$\mu_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\Sigma_0 = \begin{bmatrix} 0.01 & 0 & 0 \\ 0 & 0.01 & 0 \\ 0 & 0 & 10000 \end{bmatrix}$$

Loading [MathJax]/jax/output/HTML-CSS/jax.js

In [4]:

```
delta = 0.1
X_max = 3
Y_max = 3
X_min = -3
Y_min = -3
Theta_max = 2 * math.pi
Theta_min = 0

X = np.arange(X_min, X_max, delta)
Y = np.arange(Y_min, Y_max, delta)
Theta = np.arange(Theta_min, Theta_max, delta)
P_0=np.zeros((X.size,Y.size,Theta.size))

X_mesh, Y_mesh = np.meshgrid(X,Y)

mu = np.mat([[0.0], [0.0],[0.0]])
Sigma_0 = np.mat([[0.01, 0,0], [0,0.01,0], [0,0,10000]])

sum = 0
for i in range(X.size):
    for j in range(Y.size):
        for k in range(Theta.size):
            x = np.mat([[X[i]],[Y[j]],[Theta[k]]])
            P_0[i,j,k] = math.pow(math.e, -1.0/2.0*(x - mu).T*Sigma_0.I*(x-mu))
            sum += P_0[i,j,k]
P_0 = P_0/sum

P_0_s = np.zeros((X.size, Y.size))
for i in range(X.size):
    for j in range(Y.size):
        for k in range(Theta.size):
            P_0_s[i,j] += P_0[i,j,k]
            sum += P_0[i,j,k]
P_0_s /= sum

fig=plt.figure()
ax=fig.add_subplot(111)
ax.contourf(X_mesh, Y_mesh, P_0_s, cmap='rainbow')
plt.show()
```



Loading [MathJax]/jax/output/HTML-CSS/jax.js

b). 实现直方图滤波预测步骤并与EKF的结果进行对比

首先需要获得状态转移函数的概率分布，题目中假设机器人可以不受噪声影响的移动，那么有

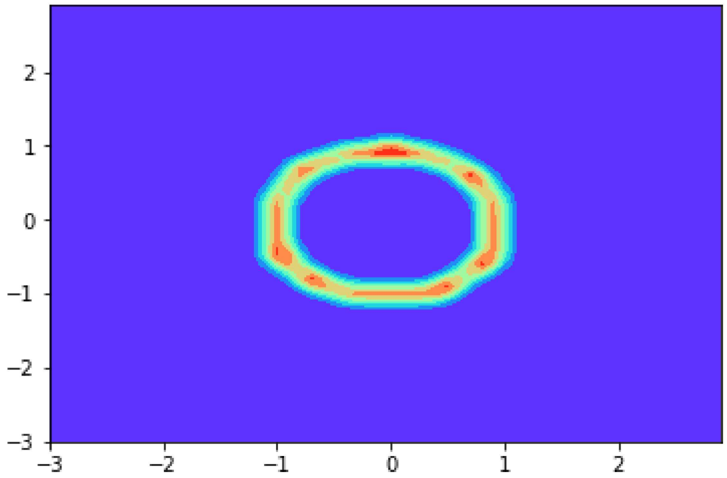
$$p(x_t|x_{t-1}) = \begin{cases} 1 & x_t = g(x_{t-1}, u_t) \\ 0 & x_t \neq g(x_{t-1}, u_t) \end{cases}$$

其中$g(x_{t-1}, u_t)$ 为预测方程

$$g(x_{t-1}, u_t) = \begin{bmatrix} x_{t-1} + u_t \cos(\theta_{t-1}) \\ y_{t-1} + u_t \sin(\theta_{t-1}) \\ \theta_{t-1} \end{bmatrix}$$

In [5]:

```
P_1 = np.zeros(P_0.shape)
delta_v = np.mat([[delta],[delta],[delta]])
x_min_v = np.mat([[X_min],[Y_min],[Theta_min]])

def g(x_t_1):
    x_t = x_t_1 + np.mat([[math.cos(x_t_1[2])],[math.sin(x_t_1[2])],[0]])
    return x_t

def compute_index(x_t):
    index = math.floor(np.div(x_t-x_min_v,delta_v))
    if index[0] >= X.size:
        index[0] = X.size -1
    if index[0] < 0:
        index[0] = 0
    if index[1] >= Y.size:
        index[1] = Y.size -1
    if index[1] < 0:
        index[1] = 0
    if index[2] >= Theta.size:
        index[2] = Theta.size -1
    if index[2] < 0:
        index[2] = 0
    return index

for i_t_1 in range(X.size):
    for j_t_1 in range(Y.size):
        for k_t_1 in range(Theta.size):
            x_t_1 = np.mat([[X[i_t_1]],[Y[j_t_1]],[Theta[k_t_1]]])
            x_t = g(x_t_1)
            i = math.floor((x_t[0] - X_min)/delta)
            j = math.floor((x_t[1] - Y_min)/delta)
            k = math.floor((x_t[2] - Theta_min)/delta)
            if i >= X.size:
                i = X.size -1
            if i < 0:
                i = 0
            if j >= Y.size:
                j = Y.size -1
            if j < 0:
                j = 0
            if k >= Theta.size:
                k = Theta.size -1
            if k < 0:
                k = 0
            P_1[i,j,k] += 1*P_0[i_t_1, j_t_1, k_t_1]

            sum = 0
P_1_s = np.zeros((X.size, Y.size))
for i in range(X.size):
    for j in range(Y.size):
        for k in range(Theta.size):
            P_1_s[i,j] += P_1[i,j,k]
            sum += P_1[i,j,k]
P_1_s /= sum

fig=plt.figure()
ax=fig.add_subplot(111)
ax.contourf(X_mesh, Y_mesh, P_1_s, cmap='rainbow')
plt.show()
```



c). 将测量归并入估计，将结果与EKF进行比较

根据题意可以知道，观测方程的噪声满足高斯分布，方差为0.01，可以得到观测方程和观测函数的概率函数如下

$$z_t = Cx_t + \delta_t$$
$$p(z_t | x_t) \sim N(Cx_t, Q_t)$$

其中 $C = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$

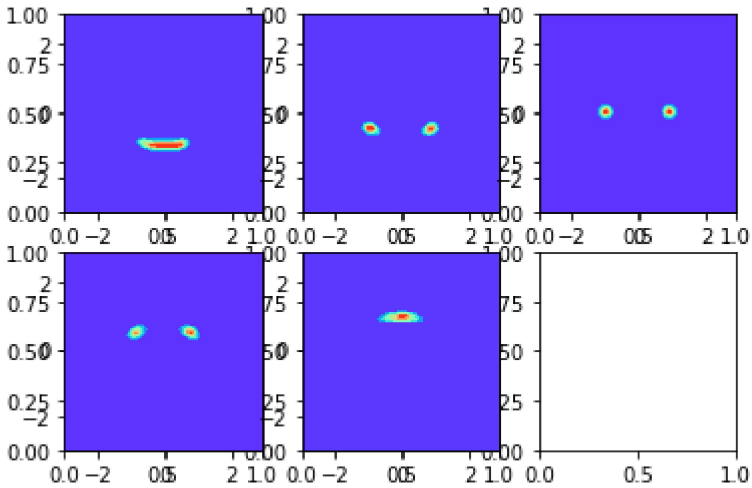
下面分别计算观测值分别为 $z = -1.0, 0.5, 0, 0.5, 1.0$ 时的位置分布，画出对应的热力图

Loading [MathJax]/jax/output/HTML-CSS/jax.js

In [6]:

```
P_1_f = np.zeros(P_1.shape)
C = np.mat([[1,0,0]])
Q = np.mat([[0.01]])

fig, ax = plt.subplots(2,3)
for fi in range(5):
    z = np.mat([[-1.0+fi*0.5]])
    print("figure", fi, "z: ", z)
    ax = fig.add_subplot(2,3,1+fi)
    for i in range(X.size):
        for j in range(Y.size):
            for k in range(Theta.size):
                x_t = np.mat([[X[i]],[Y[j]],[Theta[k]]])
                P_1_f[i,j,k] = math.pow(math.e, -1.0/2.0*(z - C*x_t).T*Q.I*(z - C*x_t))*P_1[i,j,
k]

    P_1_fs = np.zeros((X.size, Y.size))
    for i in range(X.size):
        for j in range(Y.size):
            for k in range(Theta.size):
                P_1_fs[i,j] += P_1_f[i,j,k]
            sum += P_1_fs[i,j]
    P_1_fs /= sum
    ax.contourf(X, Y, P_1_fs, cmap='rainbow')

plt.show()
```

```
figure 0 z:  [[-1.]]
figure 1 z:  [[-0.5]]
figure 2 z:  [[0.]]
figure 3 z:  [[0.5]]
figure 4 z:  [[1.]]
```



结果有些奇怪，应该是按照x来分布的，但结果像是按照y分布的，好像是坐标轴选错出了问题，还没有搞清楚是怎么回事?

Loading [MathJax]/jax/output/HTML-CSS/jax.js

3. 本章讨论了使用单一粒子的效果，如果M=2时会怎样？

采用单一粒子时，测量概率对更新结果不起作用，这是由于无论权重因子多大，都会在归一化步骤将其归一称为1，那么测量更新就唯一地由预测概率$\overline{bel}(x_t|u_t)$ 决定。

那么当$M=2$时，不考虑重采样过程导致的粒子缺乏，单一粒子时的影响会极大程度地减弱，因为此时更新测量开始起作用，不在唯一地由预测概率$\overline{bel}(x_t|u_t)$决定。但采样偏差是依然存在的，假设观测更新满足正态分布，如果两个粒子全部落在正态分布的尾部，那么下一次进行预测更新时，机器人的均值将偏离正态分布的中心。

4. 使用粒子滤波实现习题4.1

a). 不考虑观测，计算$t=1,2,...,5$时刻的概率分布

为方便推导和阅读，这里再次重申初值，方差，状态转移方程以及概率方程

$$
\mu_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \qquad \Sigma_0 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}
$$

$$
x_t = Ax_{t-1} + Bu_t + \epsilon_t
$$

$$
p(x_t|x_{t-1}) \sim N(x_t - Ax_{t-1} - Bu_t, R_t)
$$

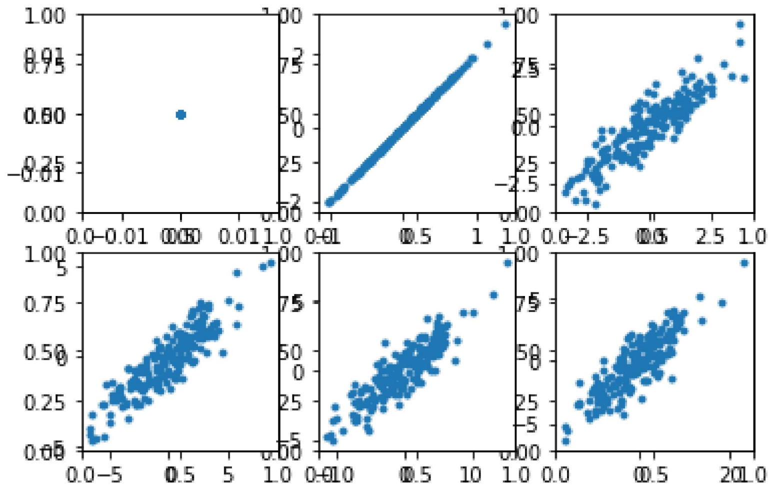
$$
A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \qquad B = \begin{bmatrix} 1/2 \\ 1 \end{bmatrix} \qquad R_t = \begin{bmatrix} 1/4 & 1/2 \\ 1/2 & 1 \end{bmatrix}
$$

如果不考虑观测步骤，各个粒子的重要性由$p(x_t|x_tt_1, u_t)$，于是可以省略重要性重采样步骤

Loading [MathJax]/jax/output/HTML-CSS/jax.js

In [57]:

```
M=200
mu_0 = np.array([0,0])
A=np.mat([[1.0, 1.0],[0.0, 1.0]])
B=np.mat([[0.5],[1.0]])
R=np.mat([[0.25, 0.5],[0.5, 1.0]])+np.ones(2)*1e-5
particles=np.zeros((M,2))
weight=np.zeros((M,1))

fig, ax = plt.subplots(2,3)
ax = fig.add_subplot(2,3,1)
ax.scatter(particles[:,0], particles[:,1], marker='.')
for fi in range(5):
    ax = fig.add_subplot(2,3,2+fi)
    for i in range(M):
        particle = np.mat(particles[i,:])
        temp_mu = A*particle.T
        temp_mu = np.array([temp_mu[0,0], temp_mu[1,0]])
        particles[i,:] = np.random.multivariate_normal(mean=temp_mu, cov=R, size=1)
    ax.scatter(particles[:,0], particles[:,1], marker='.')

plt.show()
```



b). 对观测步骤进fangc行粒子滤波，并将滤波前和滤波后结果进行对比

得到观测方程和观测概率 $p(z_t|x_t)$

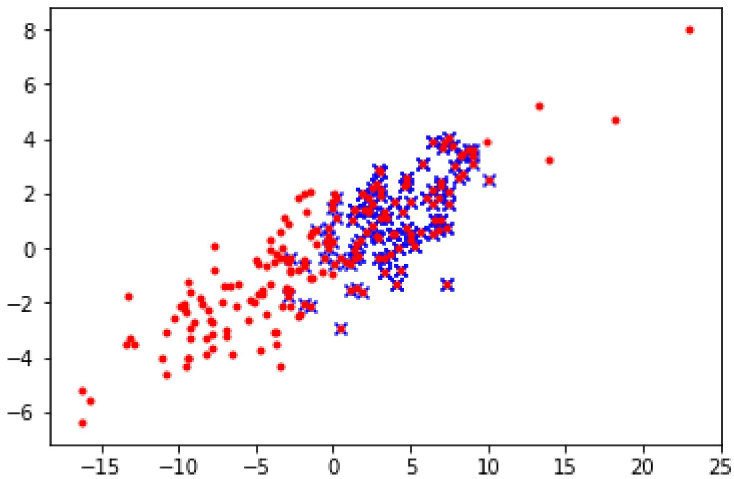
$$z_t = Cx_t + \delta_t$$
$$p(z_t|x_t) \sim N(Cx_t, Q_t)$$
$$C = \begin{bmatrix} 1 & 0 \end{bmatrix} \qquad Q_t = 10$$

首先计算各个粒子的权重，然后使用地方差采样法进行重要性重采样

In [64]:

```
C = np.mat([1,0])
Q = np.mat([10.0])
weights = np.zeros((M,1))
Weights = np.zeros((M+1,1))
z = 5

for i in range(M):
    particle = np.mat(particles[i,:])
    temp_z = C*particle.T
    weights[i,:] = math.pow(math.e, -1.0/2.0 * (z-temp_z).T*Q.I*(z-temp_z))
    Weights[i+1, :] = Weights[i,:] + weights[i,:]

particles_f=np.zeros((M,2))
delta_w = Weights[M,0]/M
j = 0
for i in range(M):
    weight = delta_w * i
    while weight > Weights[j]:
        j+=1
    particles_f[i,:]=particles[j-1,:]

fig = plt.figure()
ax = fig.add_subplot(111)
ax.scatter(particles_f[:,0], particles_f[:,1], c='b', marker='x')
ax.scatter(particles[:,0], particles[:,1], c='r', marker='.')
plt.show()
```

0.3522311005654212



如上图，蓝色叉为滤波后的分布，红色圆点为滤波前的分布

Loading [MathJax]/jax/output/HTML-CSS/jax.js

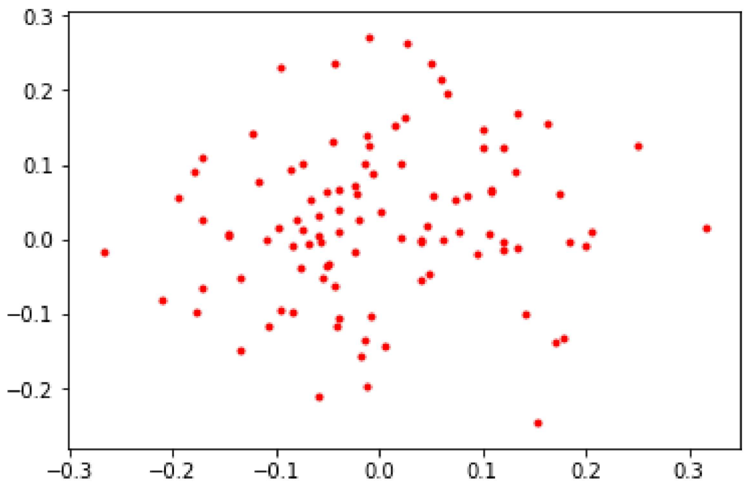
5. 使用粒子滤波实现习题4.2

a). 给出一个合理的初值估计

初值估计为正态分布，$\mu = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$ 方差 $\Sigma = \begin{bmatrix} 0.01 & 0 & 0 \\ 0 & 0.01 & 0 \\ 0 & 0 & 10000 \end{bmatrix}$

In [65]:

```
M = 100
mu_0=np.array([0,0,0])
Sigma_0=np.mat([[0.01, 0,0],[0,0.01,0],[0,0,10000]])
particles = np.random.multivariate_normal(mean=mu_0, cov=Sigma_0, size=M)

fig = plt.figure()
ax = fig.add_subplot(111)
ax.scatter(particles[:,0], particles[:,1], c='r', marker='.')
plt.show()
```



b). 转移实现粒子滤波预测步，并与EKF结果进行对比

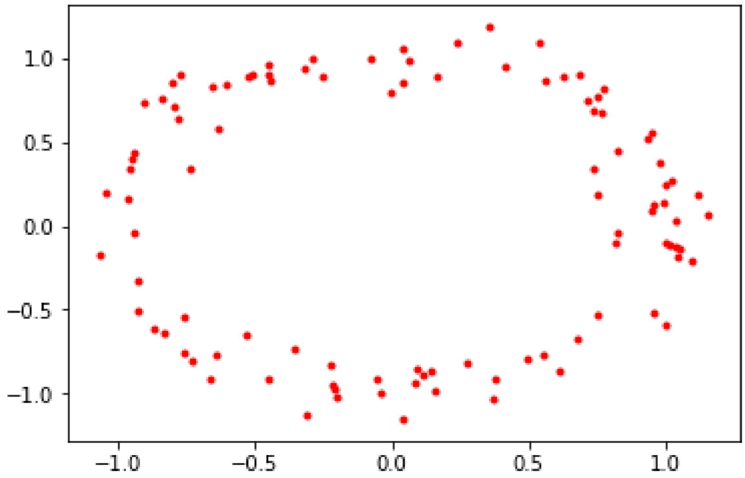
这一步假设状态转移没有噪声，也就不用从状态转移概率中采样，只需要根据状态转移函数进行计算即可，为方便计算和读者理解，这里重申状态转移函数如下

$$\begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} = \begin{bmatrix} x_{t-1} + \cos(\theta_{t-1}) \\ y_{t-1} + \sin(\theta_{t-1}) \\ \theta_{t-1} \end{bmatrix}$$

Loading [MathJax]/jax/output/HTML-CSS/jax.js

In [73]:

```
bar_particles=np.zeros(particles.shape)

def g(x_t_1):
    x_t = np.array([x_t_1[0]+math.cos(x_t_1[2]), x_t_1[1]+math.sin(x_t_1[2]), x_t_1[2]])
    return x_t

for i in range(M):
    bar_particles[i,:]=g(particles[i,:])

fig = plt.figure()
ax = fig.add_subplot(111)
ax.scatter(bar_particles[:,0], bar_particles[:,1], c='r', marker='.')
plt.show()
```



c). 考虑观测环节，并与EKF结果进行对比

每个粒子的重要性权重（即观测方程）仍然满足正态分布

$$p(z_t|x_t) \sim N(Cx_t, Q_t)$$

其中 $C = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$，方差$Q = 0.01$，假设观测值$z = -1.0, -0.5, 0.0, 0.5, 1.0$时，分别给出滤波结果

Loading [MathJax]/jax/output/HTML-CSS/jax.js

In [91]:

```
weights=np.zeros((M,1))
Weights=np.zeros((M+1,1))
particles_f=np.zeros(bar_particles.shape)
C=np.mat([1,0,0])
Q=np.mat([[0.01]])

fig, ax = plt.subplots(2,3)
plt.xlim([-2,2])
plt.ylim([-2,2])
for fi in range(5):
    z = np.mat([[-1.0+fi*0.5]])
    ax = fig.add_subplot(2,3, 1+fi)
    for i in range(M):
        temp_x=np.mat([[bar_particles[i,0]],[bar_particles[i,1]],[bar_particles[i,2]]])
        weights[i,0] = math.pow(math.e, -1.0/2.0 * (z - C*temp_x).T*Q.I*(z - C*temp_x))
        Weights[i+1,0] = Weights[i,0]+weights[i,0]

    delta_w = Weights[M,0]/M
    j = 0
    for i in range(M):
        weight = delta_w*i
        while weight > Weights[j,0]:
            j+=1
        particles_f[i,:] = bar_particles[j-1,:]
    ax.scatter(particles_f[:,0], particles_f[:,1], marker='.')


plt.show()
```