1. let $dx_k = x_k - \hat{x}_{k|k-1}$ $\quad dz_k = z_k - h_k(\hat{x}_{k|k-1})$

linearize $z_k = h_k(x_k) + v_k$, we can have:

$$z_k \approx h_k(\hat{x}_{k|k-1}) + \partial h_k(\hat{x}_{k|k-1})(x - \hat{x}_{k|k-1}) + v_k$$

let $H_k = \partial h_k(\hat{x}_{k|k-1})$

Then we have: $z_k - h_k(x) \approx \underbrace{z_k - h_k(\hat{x}_{k|k-1})} - \partial h_k(\hat{x}_{k|k-1})(x - \hat{x}_{k|k-1})$

$$= \quad dz_k \quad - \quad H_k dx_k$$

So: for $J(x)$:

$$J(x) = \frac{1}{2} dx_k^T P_{k|k-1}^{-1} dx_k + \frac{1}{2}(dz_k - H_k dx_k)^T R_k^{-1}(dz_k - H_k dx_k)$$

we want: $\partial J = P_{k|k-1}^{-1} dx_k + H_k^T R_k^{-1} H_k dx_k - H_k^T R_k^{-1} dz_k = 0$

so $\left(P_{k|k-1}^{-1} + H_k^T R_k^{-1} H_k\right) dx_k = H_k^T R_k^{-1} dz_k$

so $x_k - \hat{x}_{k|k-1} = \left(P_{k|k-1}^{-1} + H_k^T R_k^{-1} H_k\right)^{-1} H_k^T R_k^{-1} dz_k$

according to matrix inverse lemma:

$$AB[C - DAB]^{-1} = [A^{-1} - BC^{-1}D]^{-1} BC^{-1}$$

$$(P_{k|k-1}^{-1} + H_k^T R_k^{-1} H_k)^{-1} H_k^T R_k^{-1} = P_{k|k-1} H_k^T (H_k P_{k|k-1} H_k^T + R_k)^{-1} = K_k$$

for minimum, we need $\nabla^2 J > 0$

$$\nabla^2 J = P_{k|k-1}^{-1} + H_k^T R_k^{-1} H_k, \qquad P_{k|k-1}^{-1} > 0, \quad H_k^T R_k^{-1} H_k > 0$$

so $\nabla^2 J > 0$

So for the least square approach mentioned EKF correction would be:

$$x = \hat{x}_{k|k-1} + K_k[z_k - h(\hat{x}_{k|k-1})]$$

$$K_k = P_{k|k-1} H_k^T (H_k P_{k|k-1} H_k^T + R_k)^{-1}.$$

2)
$$\begin{bmatrix}\dot{\theta}\\ \dot{\beta}\end{bmatrix} = \begin{bmatrix}0 & -1\\ 0 & 0\end{bmatrix}\begin{bmatrix}\theta\\ \beta\end{bmatrix} + \begin{bmatrix}1\\ 0\end{bmatrix}w + \begin{bmatrix}-1 & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}\eta_v\\ \eta_u\end{bmatrix}$$

let $F = \begin{bmatrix}0 & -1\\ 0 & 0\end{bmatrix}$ $G = \begin{bmatrix}1\\ 0\end{bmatrix}$ $L = \begin{bmatrix}-1 & 0\\ 0 & 1\end{bmatrix}$

then, $\Phi_k = e^{\Delta t F} = \begin{bmatrix}1 & -\Delta t\\ 0 & 1\end{bmatrix}$ $\Gamma_k = \int_{t_k}^{t_{k+1}} \Phi_k \cdot G(\tau) d\tau = \begin{bmatrix}\Delta t\\ 0\end{bmatrix}$

$$Q_k = \int_{t_k}^{t_{k+1}} \Phi(t_{k+1}, \tau) \cdot L(\tau) Q_c' \delta(\tau - \alpha) L(\alpha)^T \cdot \Phi^T(t_k, \alpha) d\tau d\alpha$$

$$= \int_{t_k}^{t_{k+1}} \begin{bmatrix}1 & \tau - t_{k+1}\\ 0 & 1\end{bmatrix}\begin{bmatrix}-1 & 0\\ 0 & 1\end{bmatrix} Q_c' \begin{bmatrix}-1 & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0\\ \tau - t_{k+1} & 1\end{bmatrix} d\tau$$

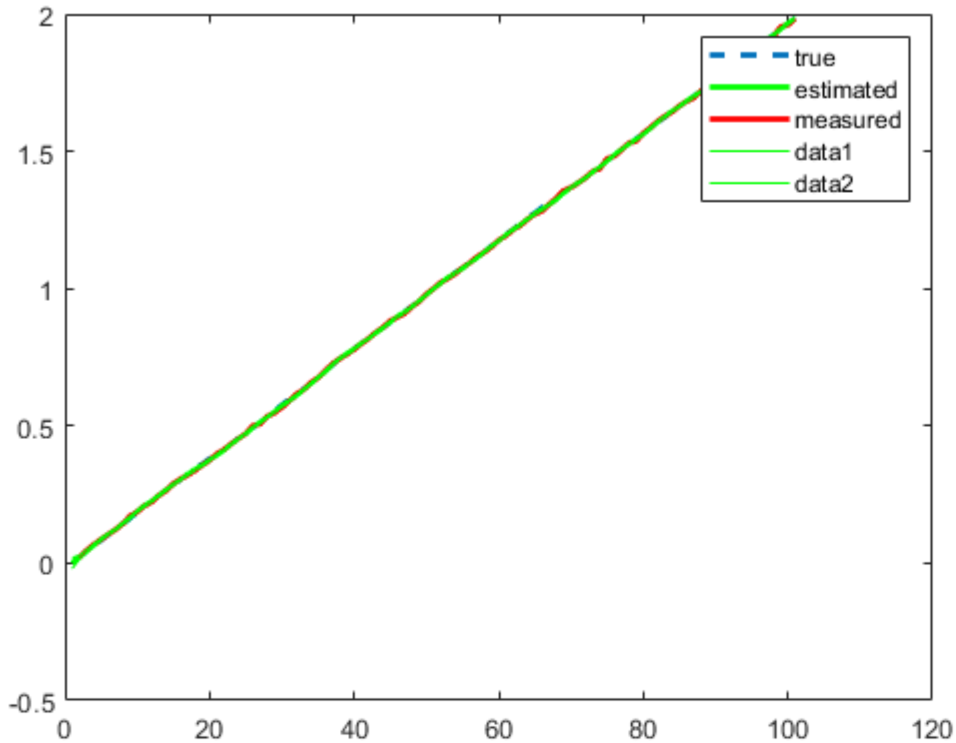
$$Q_c' = \begin{bmatrix}\sigma_v^2 & 0\\ 0 & \sigma_u^2\end{bmatrix}$$

$$= \int_{t_k}^{t_{k+1}} \begin{bmatrix}\sigma_v^2 + \sigma_u^2(\tau - t_{k+1})^2 & \sigma_u^2(\tau - t_{k+1})\\ \sigma_u^2(\tau - t_{k+1}) & \sigma_u^2\end{bmatrix} d\tau$$

$$= \begin{bmatrix}\sigma_v^2 \cdot \Delta t + \frac{1}{3}\sigma_u^2 \cdot \Delta t^3 & -\frac{1}{2}\sigma_u^2 \Delta t^2\\ -\frac{1}{2}\sigma_u^2 \Delta t^2 & \sigma_u^2 \cdot \Delta t\end{bmatrix}$$

```
clc;
clear;
% Kalman filtering of the double integrator with position measurements
% timing
dt = 1; % time-step
N = 100; % total time-steps
T = N*dt; % final time

% noise terms
sigma_n = 1.5e-5;
sigma_u = 3e-9;
sigma_v = 3e-6;
S.q = diag([sigma_u,sigma_v]); % external disturbance variance

S.r = sigma_n; % measurement noise variance

% F matrix
S.F = [1 -dt;
       0 1]

% A matrix
S.A = [-dt, -dt^2/2;0, dt];

% G matrix
S.G = [dt;0];

% Q matrix
S.Q = [sigma_v*dt + (1/3)*sigma_u*(dt)^3, -0.5*sigma_u*(dt)^2;
-0.5*sigma_u*(dt)^2, sigma_u*dt];

% R matrix
S.R = S.r;

% H
S.H = [1, 0];

% initial estimate of mean and covariance
x = [0; 1.7e-7];
P = diag([1e-4 1e-12]);
xts = zeros(2, N+1); % true states
xs = zeros(2, N+1); % estimated states
Ps = zeros(2, 2, N+1); % estimated covariances
zs = zeros(1, N); % estimated state
pms = zeros(1, N); % measured position

yita_u = sqrt(sigma_u)*randn(1);
betas = yita_u*(1:N); %bias of u

xts(:,1) = x;
xs(:,1) = x;
Ps(:,:,1) = P;
```

```
for k=1:N
    yita_v = sqrt(sigma_v)*randn(1);
    u = 0.02 + betas(k) + yita_v;
    xts(:,k+1) = S.F*xts(:,k) + S.G*u; % + sqrt(S.q)*randn; % true state

    [x,P] = kf_predict(x,P,u,S); % prediction
    z = xts(1,k+1) + sqrt(S.r)*randn; % generate random measurement

    [x,P] = kf_correct(x,P,z,S); % correction
    % record result
    xs(:,k+1) = x;
    Ps(:,:,k+1) = P;
    zs(:,k) = z;
end

plot(xts(1,:), '--', 'LineWidth',2)
hold on
plot(xs(1,:), 'g', 'LineWidth',2)
plot(2:N+1,zs(1,:), 'r', 'LineWidth',2)
legend('true', 'estimated','measured')

plot(xs(1,:) + 1.96*reshape(sqrt(Ps(1,1,:)),N+1,1)', '-g')
plot(xs(1,:) - 1.96*reshape(sqrt(Ps(1,1,:)),N+1,1)', '-g')

function [x,P] = kf_predict(x, P, u, S)
    x = S.F*x + S.G*u;
    P = S.F*P*S.F' + S.Q;
end

function [x,P] = kf_correct(x, P, z, S)
    K = P*S.H'*inv(S.H*P*S.H' + S.R);
    P = (eye(length(x)) - K*S.H)*P;
    x = x + K*(z - S.H*x);
end

S =
  ####### struct:

    q: [2×2 double]
    r: 1.5000e-05
    F: [2×2 double]
```


true
estimated
measured
data1
data2

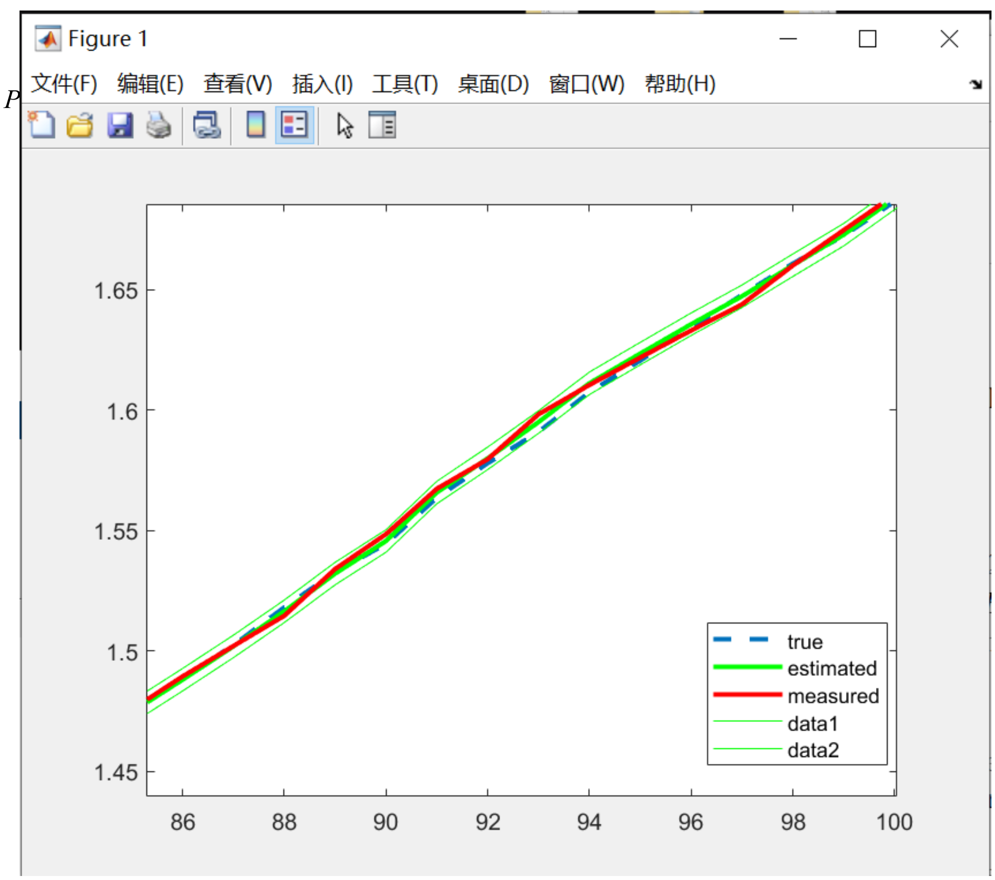


Figure 1
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
true
estimated
measured
data1
data2

3) Jacobian F is :

$$F = \partial_x f = \begin{bmatrix} 1 & 0 & -\Delta t \cdot r \Omega \sin\theta & \Delta t \Omega \cos\theta \\ 0 & 1 & \Delta t \, r \Omega \cos\theta & \Delta t \Omega \sin\theta \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
clc;
clear;
%rng('default')
rng(10212);
S.bearing_only = 0;

% two beacons at (-2,2) and (2,2) : system is observable (two or more)
S.pbs = [-2, 2;
        2, 2]; % beacon positions
nb = size(S.pbs,2); % number of beacons
if S.bearing_only
    S.h = @b_h; % bearing sensing
    S.r = nb; % measurement dimension
    S.R = .4*diag(repmat([.1], nb, 1));
else
    S.h = @br_h; % bearing-reange sensing
    S.r = 2*nb; % measurement dimension
    S.R = .4*diag(repmat([.1; .01], nb, 1));
end

S.n = 4; % state dimension
S.f = @uni_f; % mobile-robot dynamics


% timing
dt = .1;
%N = 2580;
N = 50;
T = dt*N;
S.dt = dt;
% noise models
S.Q = dt^2*diag([.01 .01 .01 .0001]);
% initial mean and covariance
xt = [0; 0; 0; 1]; % true state

P = diag([0.01 0.01 0.01 0.04]); % covariance

x = xt + sqrt(P)*randn(S.n, 1); % initial estimate with added noise
xts = zeros(S.n, N+1); % true states
xs = zeros(S.n, N+1); % estimated states
Ps = zeros(S.n, S.n, N+1); % estimated covariances
ts = zeros(N+1,1); % times

zs = zeros(S.r, N); % measurements
xts(:, 1) = xt;
xs(:, 1) = x;
Ps(:, :, 1) = P;
ts(1) = 0;

ds = zeros(S.n, N+1); % errors
ds(:,1) = x - xt;
```

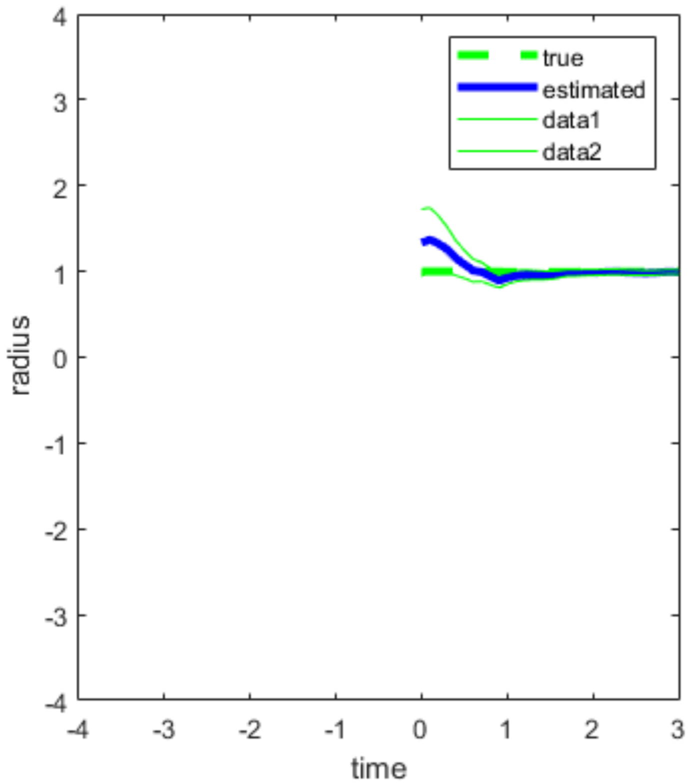
```
for k=1:N
    u = dt*[2; 1]; % known controls
    xts(:,k+1) = S.f(xts(:,k), u, S) + sqrt(S.Q)*randn(4,1); % true state
    [x,P] = ekf_predict(x, P, u, S); % predict
    ts(k+1) = k*dt;
    z = S.h(xts(:,k+1), S) + sqrt(S.R)*randn(S.r,1); % generate measurement
    [x,P] = ekf_correct(x, P, z, S); % correct
    xs(:,k+1) = x;
    Ps(:,:,k+1) = P;
    zs(:,k) = z;
    ds(:,k+1) = x - xts(:,k+1); % actual estimate error
    ds(:,k+1) = fix_state(ds(:,k+1));
end

fig1=figure
fig2=figure
fig3=figure
for k=1:N
    figure(fig1)
    plot(ts(1:k), xts(4,1:k), '--g','LineWidth',3)
    hold on
    plot(ts(1:k), xs(4,1:k), '-b','LineWidth',3)
    legend('true', 'estimated')
    hold on

    % 95% confidence intervals of the estimated position
    p_s = reshape(Ps(4,4,1:k),1,k);

    plot(ts(1:k), xs(4,1:k) + 1.96*sqrt(p_s), '-g')
    plot(ts(1:k), xs(4,1:k) - 1.96*sqrt(p_s), '-g')

    xlabel('time')
    ylabel('radius')
    axis equal
    axis xy
    axis([-4 3 -4 4])

    figure(fig2)
    subplot(2,1,1)
    hold off
    plot(ds(:,1:k)')

    xlabel('k')
    ylabel('meters or radians')
    legend('e_x','e_y','e_\theta','e_r')
    subplot(2,1,2)
    hold off

    plot(ts(1:k), reshape(sqrt(Ps(1,1,1:k)),k,1), ...
    ts(1:k), reshape(sqrt(Ps(2,2,1:k)),k,1), ...
    ts(1:k), reshape(sqrt(Ps(3,3,1:k)),k,1), ...
    ts(1:k), reshape(sqrt(Ps(4,4,1:k)),k,1));
    legend('\sigma_x','\sigma_y','\sigma_\theta','\sigma_r')
    xlabel('t')
```

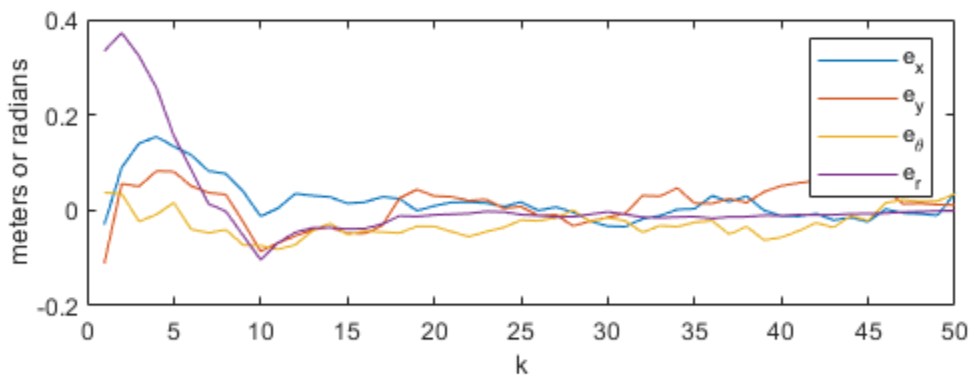
```
    ylabel('meters or radians')
    figure(fig3)
    plot(xts(1,1:k), xts(2,1:k), '--g','LineWidth',3)
    hold on
    plot(xs(1,1:k), xs(2,1:k), '-b','LineWidth',3)
    legend('true', 'estimated')

    % beacon
    plot(S.pbs(1,:), S.pbs(2,:), '*r');
    plotcov2(xs(1:2,k), 1.96^2*Ps(1:2,1:2,k));

    xlabel('x')
    ylabel('y')
    axis equal
    axis xy
    axis([-4 3 -4 4])
    drawnow
    if k==1

    end
end


function [x, varargout] = uni_f(x, u, S)
% dynamical model of the unicycle
    c = cos(x(3));
    s = sin(x(3));
    x = [x(1) + c*u(1)*x(4);
    x(2) + s*u(1)*x(4);
    x(3) + u(2);
    x(4)];
    x = fix_state(x, S);
    if nargout > 1
    % F-matrix
        varargout{1} = [1, 0, -s*u(1)*x(4),u(1)*c;
        0, 1, c*u(1)*x(4),u(1)*s;
        0 0 1 0;
        0 0 0 1];
    end
end

function [y, varargout] = br_h(x, S)
    p = x(1:2);
    y = [];
    H = [];
    for i=1:size(S.pbs, 2)
        pb = S.pbs(:, i); %i-th beacon
        d = pb - p;
        r = norm(d);
        th = fangle(atan2(d(2), d(1)) - x(3));
        y = [y; th; r];
        if nargout > 1
        % H-matrix
        H = [H;
```

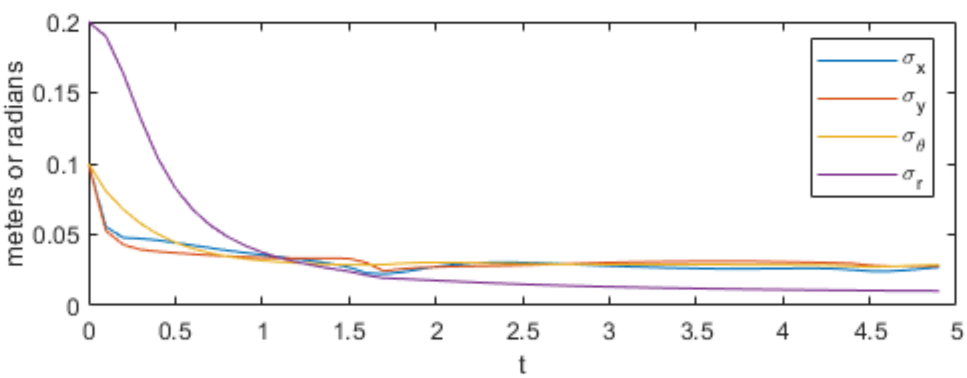
```
        d(2)/r^2, -d(1)/r^2, -1, 0;
        -d'/r, 0, 0];
        end
    end

    if nargout > 1
        varargout{1} = H;
    end
end

function [y, varargout] = b_h(x, S)
    p = x(1:2);
    y = [];
    H = [];
    for i=1:size(S.pbs, 2)
        pb = S.pbs(:, i); %i-th beacon
        d = pb - p;
        r = norm(d);
        th = fangle(atan2(d(2), d(1)) - x(3));
        y = [y; th];
        if nargout > 1
        % H-matrix
            H = [H;
                d(2)/r^2, -d(1)/r^2, -1];
        end
    end
    if nargout > 1
        varargout{1} = H;
    end
end

function [x,P] = ekf_predict(x, P, u, S)
    [x, F] = S.f(x, u, S);
    x = fix_state(x, S); % fix any [-pi,pi] issues
    P = F*P*F' + S.Q;
end

function [x,P] = ekf_correct(x, P, z, S)
    [y, H] = S.h(x, S);
    P = P - P*H'*inv(H*P*H' + S.R)*H*P;
    K = P*H'*inv(S.R);
    e = z - y;
    e = fix_meas(e, S); % fix any [-pi,pi] issues
    x = x + K*e;
end


function x = fix_state(x, S)
    x(3) = fangle(x(3));
end

function z = fix_meas(z, S)
    for i=1:size(S.pbs,2)
        if S.bearing_only
```

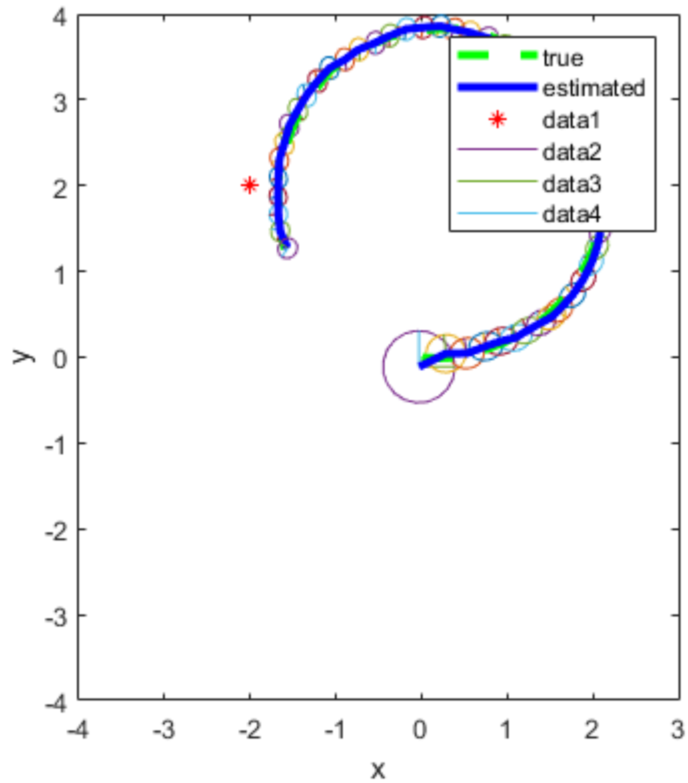
```
            z(i) = fangle(z(i));
        else
            z(2*i-1) = fangle(z(2*i-1));
        end
    end
end

function a = fangle(a)
% make sure angle is between -pi and pi
    a = mod(a,2*pi);
    if a < -pi
        a = a + 2*pi;
    else
        if a > pi
            a = a - 2*pi;
        end
    end
end
```

```
fig1 =
  Figure (1) - ##:

      Number: 1
        Name: ''
       Color: [0.9400 0.9400 0.9400]
    Position: [1000 918 560 420]
       Units: 'pixels'

  ## GET ######
fig2 =
  Figure (2) - ##:

      Number: 2
        Name: ''
       Color: [0.9400 0.9400 0.9400]
    Position: [1000 918 560 420]
       Units: 'pixels'

  ## GET ######
fig3 =
  Figure (3) - ##:

      Number: 3
        Name: ''
       Color: [0.9400 0.9400 0.9400]
    Position: [1000 918 560 420]
       Units: 'pixels'

  ## GET ######
##: ##########
```


true
estimated
data1
data2
radius
time


e_x
e_y
e_θ
e_r
meters or radians
k


σ_x
σ_y
σ_θ
σ_r
meters or radians
t



Published with MATLAB® R2021b

```
function f = uni_ekf_test2
clc;
clear
%rng('default')
rng(10212)
S.bearing_only = 1;
% single beacon at (-2,2) : system is unobservable
%S.pbs = [-2;
% 2]; % beacon positions
% two beacons at (-2,2) and (2,2) : system is observable (two or more)
S.pbs = [-2, 2;
2, 2]; % beacon positions
nb = size(S.pbs,2); % number of beacons
if S.bearing_only
    S.h = @b_h; % bearing sensing
    S.r = nb; % measurement dimension
    S.R = .4*diag(repmat([.1], nb, 1));
else
    S.h = @br_h; % bearing-reange sensing
    S.r = 2*nb; % measurement dimension
    S.R = .4*diag(repmat([.1; .01], nb, 1));
end
S.n = 4; % state dimension
S.f = @uni_f; % mobile-robot dynamics
% timing
dt = .1;
%N = 2580;
N = 50;
T = dt*N;
S.dt = dt;
% noise models
S.Q = dt^2*diag([.01 .01 .01 .0001]);
% initial mean and covariance
xt = [0; 0; 0; 1]; % true state
P = diag([0.01 0.01 0.01 0.04]); % covariance

x = xt + sqrt(P)*randn(S.n, 1); % initial estimate with added noise
xts = zeros(S.n, N+1); % true states
xs = zeros(S.n, N+1); % estimated states
Ps = zeros(S.n, S.n, N+1); % estimated covariances
ts = zeros(N+1,1); % times
zs = zeros(S.r, N); % measurements
xts(:, 1) = xt;
xs(:, 1) = x;
Ps(:, :, 1) = P;
ts(1) = 0;
ds = zeros(S.n, N+1); % errors
ds(:,1) = x - xt;

for k=1:N
    u = dt*[2; 1]; % known controls
    xts(:,k+1) = S.f(xts(:,k), u, S) + sqrt(S.Q)*randn(4,1); % true state
```

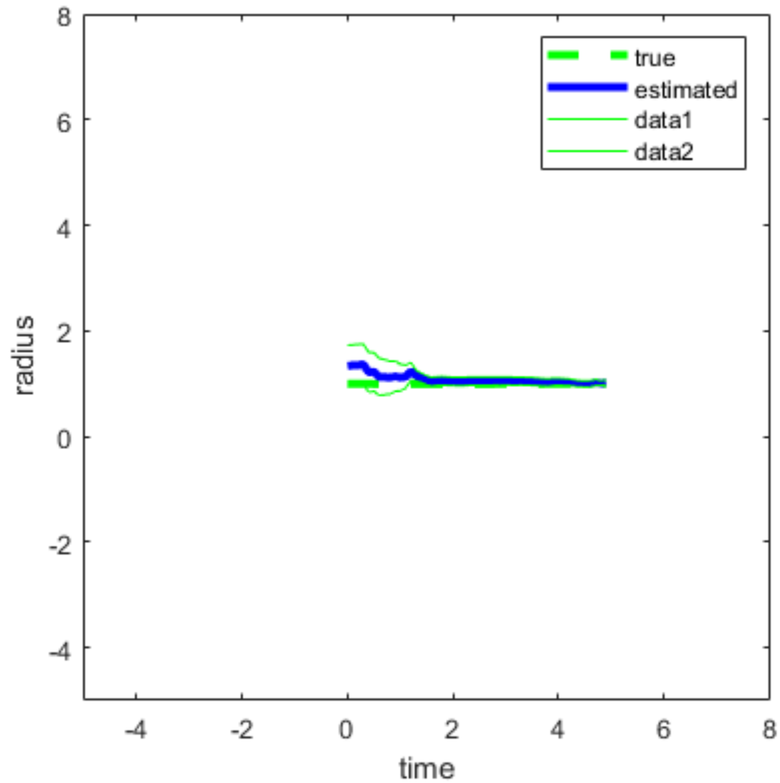
```
    [x,P] = ekf_predict(x, P, u, S); % predict
    ts(k+1) = k*dt;

    z = S.h(xts(:,k+1), S) + sqrt(S.R)*randn(S.r,1); % generate measurement
    [x,P] = ekf_correct(x, P, z, S); % correct
    xs(:,k+1) = x;
    Ps(:,:,k+1) = P;
    zs(:,k) = z;
    ds(:,k+1) = x - xts(:,k+1); % actual estimate error
    ds(:,k+1) = fix_state(ds(:,k+1));
end

fig1=figure
fig2=figure
fig3=figure

for k=1:N
    figure(fig1)
    plot(ts(1:k), xts(4,1:k), '--g','LineWidth',3)
    hold on
    plot(ts(1:k), xs(4,1:k), '-b','LineWidth',3)
    legend('true', 'estimated')
    hold on

    % beacon
    %plot(S.pbs(1,:), S.pbs(2,:), '*r');
    %plotcov2(xs(1,k), 1.96^2*Ps(4,4,k));
    % 95% confidence intervals of the estimated position
    p_s = reshape(Ps(4,4,1:k),1,k);
    plot(ts(1:k), xs(4,1:k) + 1.96*sqrt(p_s), '-g')
    plot(ts(1:k), xs(4,1:k) - 1.96*sqrt(p_s), '-g')

    xlabel('time')
    ylabel('radius')
    axis equal
    axis xy
    axis([-5 8 -5 8])

    figure(fig2)
    subplot(2,1,1)
    hold off
    plot(ds(:,1:k)')
    xlabel('k')
    ylabel('meters or radians')
    legend('e_x','e_y','e_\theta','e_r')
    subplot(2,1,2)
    hold off
    plot(ts(1:k), reshape(sqrt(Ps(1,1,1:k)),k,1), ...
    ts(1:k), reshape(sqrt(Ps(2,2,1:k)),k,1), ...
    ts(1:k), reshape(sqrt(Ps(3,3,1:k)),k,1), ...
    ts(1:k), reshape(sqrt(Ps(4,4,1:k)),k,1));
    legend('\sigma_x','\sigma_y','\sigma_\theta','\sigma_r')
    xlabel('t')
    ylabel('meters or radians')
```

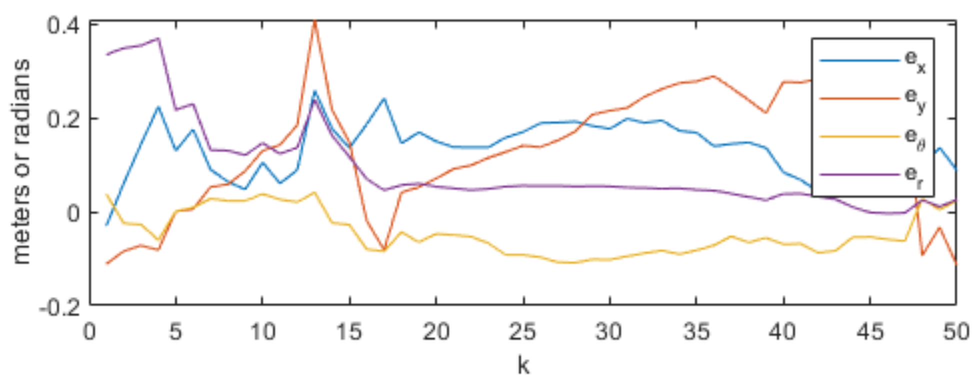
```
    figure(fig3)
    plot(xts(1,1:k), xts(2,1:k), '--g','LineWidth',3)
    hold on
    plot(xs(1,1:k), xs(2,1:k), '-b','LineWidth',3)
    legend('true', 'estimated')
    % beacon
    plot(S.pbs(1,:), S.pbs(2,:), '*r');
    plotcov2(xs(1:2,k), 1.96^2*Ps(1:2,1:2,k));

    xlabel('x')
    ylabel('y')
    axis equal
    axis xy
    axis([-5 8 -5 8])
    drawnow
    if k==1
    end
end

function [x, varargout] = uni_f(x, u, S)
% dynamical model of the unicycle
    c = cos(x(3));
    s = sin(x(3));
    x = [x(1) + c*u(1)*x(4);
    x(2) + s*u(1)*x(4);
    x(3) + u(2);
    x(4)];
    x = fix_state(x, S);
    if nargout > 1
    % F-matrix
    varargout{1} = [1, 0, -s*u(1)*x(4),u(1)*c;
    0, 1, c*u(1)*x(4),u(1)*s;
    0 0 1 0;
    0 0 0 1];
    end
end

function [y, varargout] = br_h(x, S)
    p = x(1:2);
    y = [];
    H = [];
    for i=1:size(S.pbs, 2)
        pb = S.pbs(:, i); %i-th beacon
        d = pb - p;
        r = norm(d);
        th = fangle(atan2(d(2), d(1)) - x(3));
        y = [y; th; r];
        if nargout > 1
        % H-matrix
            H = [H;
            d(2)/r^2, -d(1)/r^2, -1, 0];
        end
    end
```

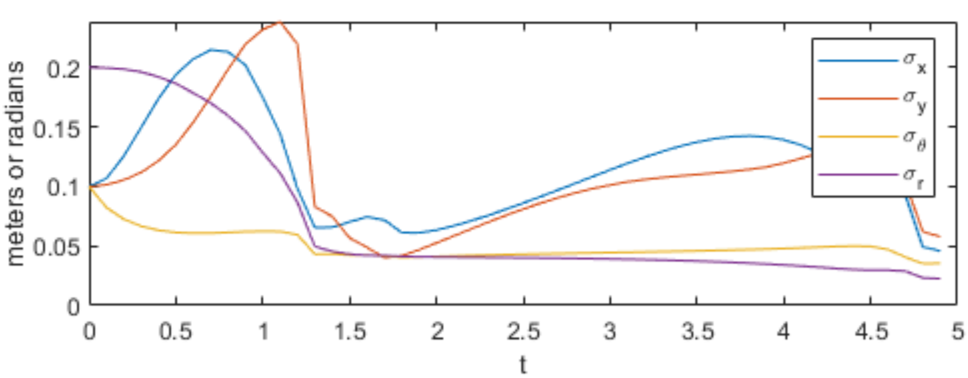
```
    if nargout > 1
        varargout{1} = H;
    end
end

function [y, varargout] = b_h(x, S)
    p = x(1:2);
    y = [];
    H = [];
    for i=1:size(S.pbs, 2)
        pb = S.pbs(:, i); %i-th beacon
        d = pb - p;
        r = norm(d);
        th = fangle(atan2(d(2), d(1)) - x(3));
        y = [y; th];
        if nargout > 1
        % H-matrix
            H = [H;
                d(2)/r^2, -d(1)/r^2, -1,0];
        end
    end
    if nargout > 1
        varargout{1} = H;
    end
end

function [x,P] = ekf_predict(x, P, u, S)
    [x, F] = S.f(x, u, S);
    x = fix_state(x, S); % fix any [-pi,pi] issues
    P = F*P*F' + S.Q;
end

function [x,P] = ekf_correct(x, P, z, S)
    [y, H] = S.h(x, S);
    P = P - P*H'*inv(H*P*H' + S.R)*H*P;
    K = P*H'*inv(S.R);
    e = z - y;
    e = fix_meas(e, S); % fix any [-pi,pi] issues
    x = x + K*e;
end

function x = fix_state(x, S)
    x(3) = fangle(x(3));
end

function z = fix_meas(z, S)

    for i=1:size(S.pbs,2)
        if S.bearing_only
            z(i) = fangle(z(i));
        else
            z(2*i-1) = fangle(z(2*i-1));
        end
    end
```

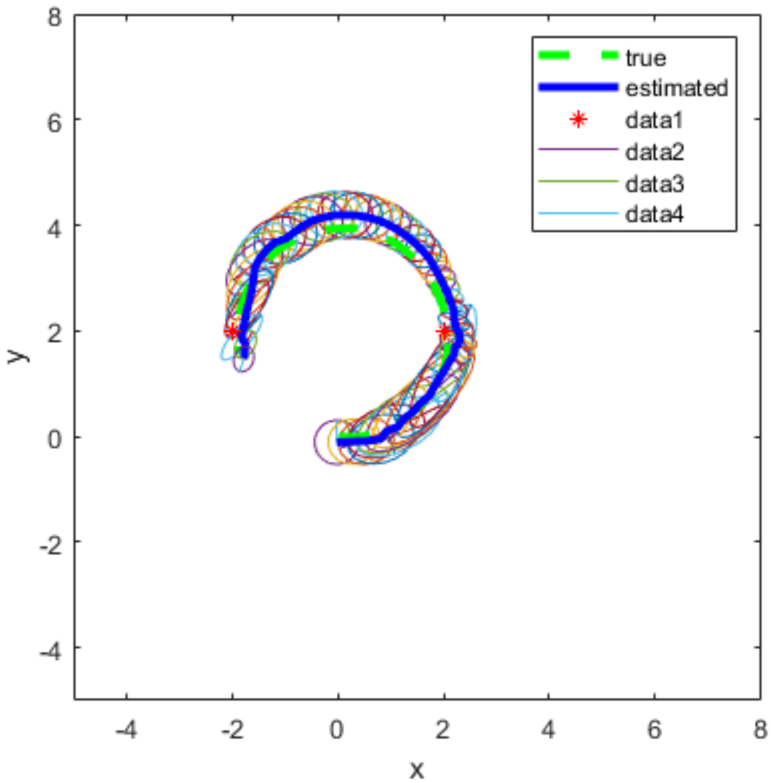
```
end

function a = fangle(a)
% make sure angle is between -pi and pi
    a = mod(a,2*pi);
    if a < -pi
        a = a + 2*pi;
    else
        if a > pi
            a = a - 2*pi;
        end
    end
end

end
```

```
fig1 =
  Figure (1) - ##:

      Number: 1
        Name: ''
       Color: [0.9400 0.9400 0.9400]
    Position: [1000 918 560 420]
       Units: 'pixels'

  ## GET ######
fig2 =
  Figure (2) - ##:

      Number: 2
        Name: ''
       Color: [0.9400 0.9400 0.9400]
    Position: [1000 918 560 420]
       Units: 'pixels'

  ## GET ######
fig3 =
  Figure (3) - ##:

      Number: 3
        Name: ''
       Color: [0.9400 0.9400 0.9400]
    Position: [1000 918 560 420]
       Units: 'pixels'

  ## GET ######
##: ##########
```


true
estimated
data1
data2
radius
time


e_x
e_y
e_θ
e_r
meters or radians
k

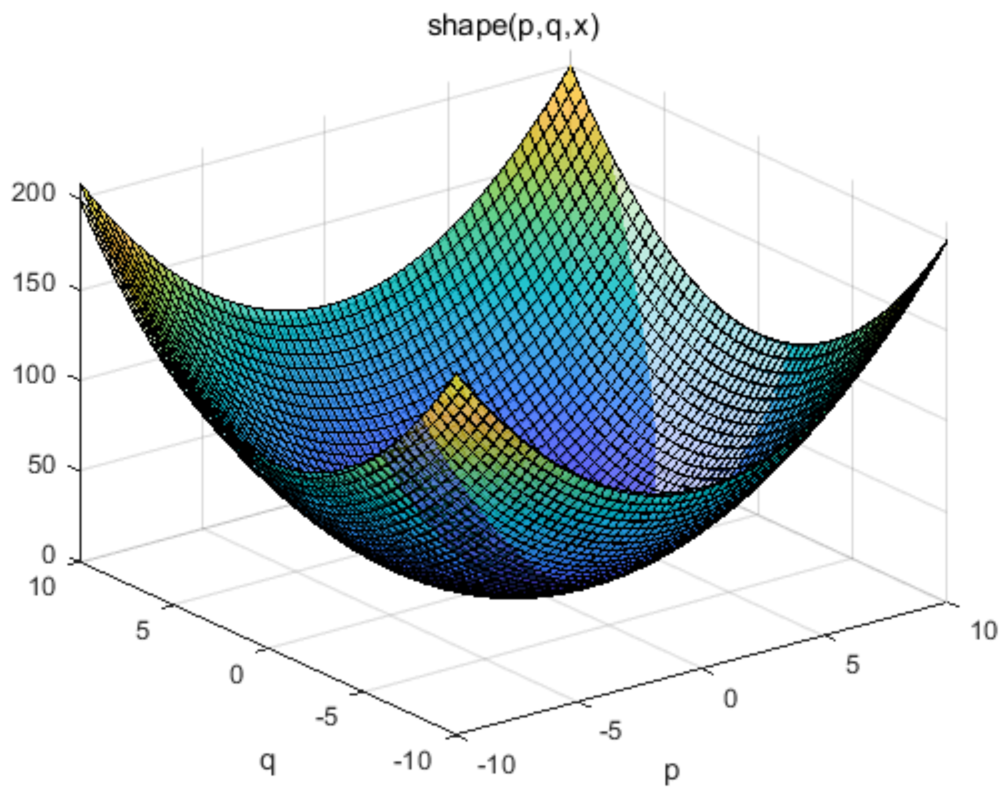

σ_x
σ_y
σ_θ
σ_r
meters or radians
t



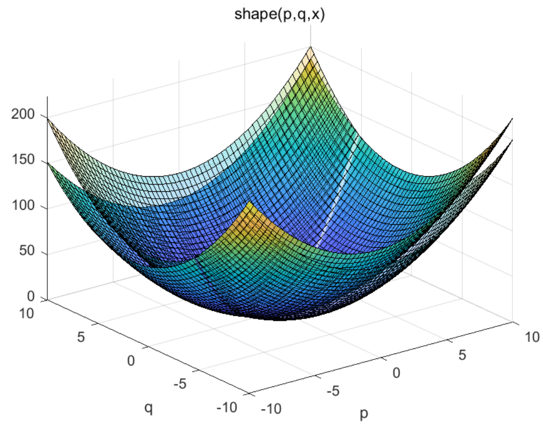
Published with MATLAB® R2021b

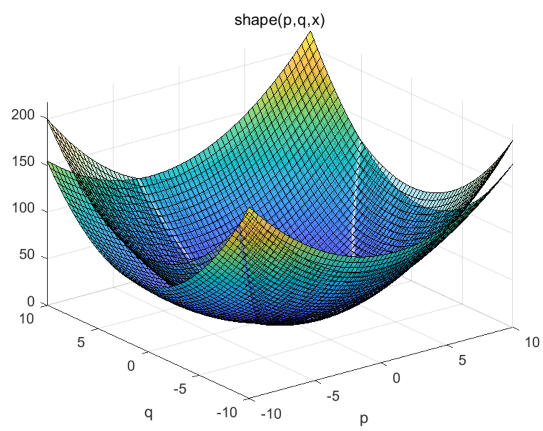
```
clc;
clear;

s = 10;
% true shape parameter (i.e. a symmetric cup)
x_true = [1; 1; 0; 0; 0; 0];

% prior parameters:
m = [1.2; 1.3; 1; 1; 1; 1];
%m = [1.2; 1.2; 1.2; 1.2; 1.2; 1.2];
%m = [1.3; 1.3; 1.3; 1.3; 1.3; 1.3];
%m = [1.2; 1.3; 1; 1; 1; 1]*2;
%m = [1.2; 1.3; 1; 1; 1; 1]*10;

P0 = diag([16,16,16,16,16,16]);

% plot true
gt = ezsurf(@(p,q)shape(p, q, x_true),[-s,s]);
alpha(gt, 0.3)
hold on
% measurement standard dev
std = 20;

% #of measurements
k = 8;
% generate random measurements
p = 4*s*(rand(k,1) - .5);
q = 4*s*(rand(k,1) - .5);
z = shape(p, q, x_true) + randn(k,1)*std;
% estimate optimal parameters x
R = diag(repmat(std^2, k, 1));

H = shape_basis(p, q);
x = m;
P = P0;
for i = 1:k/2
    start = 2*i-1;
    goal = 2*i;

    R_i = R( start : goal , start : goal );
    H_i = H(start : goal,:);
    Z_i = z(start : goal);
    P = inv(inv(P) + (H_i')*inv(R_i)*H_i);
    Ki = (P*H_i')*inv(R_i);
    x = x + Ki*(Z_i - H_i*x);
end
% plot estimated

ge = ezsurf(@(p,q)shape(p,q,x),[-s,s]);
alpha(ge, .8)

function f = shape_basis(p, q)
```

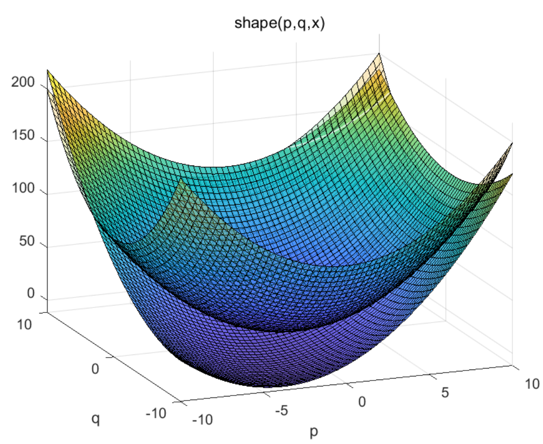
```
% quadratic function, although could be any shape
    f = [p.^2, q.^2, p.*q, p, q, ones(size(p))];
end

function z = shape(p, q, x)
    z = shape_basis(p, q)*x;
end
```



Published with MATLAB® R2021b


shape(p,q,x)
200
150
100
50
0
10
5
0
-5
-10
q
-10
-5
0
5
10
p


shape(p,q,x)
200
150
100
50
0
10
5
0
-5
-10
q
-10
-5
0
5
10
p


shape(p,q,x)
200
150
100
50
0
10
0
-10
q
-10
-5
0
5
10
p