Theodore Wilkening

ECE_6555 Final

1) Assume that $U \sim N(0,1)$ and Set $V = |U|$

note: based on the assumptions of U, I assume that U is a scalar variable

1-a) optimal estimator of V from U of the form αU, linear estimator, to minimize the MSE. Provide corresponding MSE (numerical estimate)

~~let $\hat{V} = \alpha U$~~ reference lecture 6, pg 6

note: V is not a centered variable. So set $\tilde{V} = V - \mu_V$, $\mu_V = \mathbb{E}[V] (= \mathbb{E}[|U|])$

then, $\hat{\tilde{V}} = \hat{V} - \mu_V \Rightarrow$ that is $\mathbb{E}[\tilde{V}|U] = \mathbb{E}[V|U] - \mu_V$, also now $\hat{\tilde{V}} = \alpha U$

$$\text{let } P(\alpha) = \mathbb{E}[(\tilde{V} - \hat{\tilde{V}})(\tilde{V} - \hat{\tilde{V}})] = \mathbb{E}[(\tilde{V} - \alpha U)(\tilde{V} - \alpha U)]$$
$$= \mathbb{E}[\tilde{V}^2] - 2\alpha R_{\tilde{V}U} + \mathbb{E}[\alpha^2 U^2]$$
$$= R_{\tilde{V}} - 2\alpha R_{\tilde{V}U} + \alpha^2$$

$$\frac{\partial P(\alpha)}{\partial \alpha} = -2R_{\tilde{V}U} + 2\alpha = 0 \Rightarrow \alpha = R_{\tilde{V}U} = \mathbb{E}[\tilde{V}U] = \mathbb{E}[(V - \mu_V)U] = R_{VU} - \mu_V \cancelto{0}{\mathbb{E}[U]}$$

$$\Rightarrow \boxed{\alpha = R_{VU}}$$

↑ set equal to 0 to minimize w.r.t. α

now, $\hat{\tilde{V}} = R_{VU} U$, $\hat{V} = R_{VU} U + \mu_V$

MSE
$$\mathbb{E}[(V - \hat{V})(V - \hat{V})] = \mathbb{E}[(V - R_{VU}U - \mu_V)(V - R_{VU}U - \mu_V)]$$
$$= \mathbb{E}[V^2 - 2VUR_{VU} + 2U\mu_V R_{VU} - 2V\mu_V + R_{VU}^2 U^2 + \mu_V^2]$$
$$= R_V - \cancel{2}R_{VU}^2 + 2\mu_V R_{VU} \cancelto{0}{\mathbb{E}[U]} - \cancel{2}\mu_V^2 + \cancel{R_{VU}^2} + \cancel{\mu_V^2}$$

$$\Rightarrow \boxed{MSE = R_V - R_{VU}^2 - \mu_V^2}$$

where
$R_V = \mathbb{E}[V^2] = \mathbb{E}[(\sqrt{U^2})^2]$
$R_{VU} = \mathbb{E}[VU] = \mathbb{E}[U\sqrt{U^2}]$
$\mu_V = \mathbb{E}[V] = \mathbb{E}[\sqrt{U^2}]$

See python code for estimate numerically

$\boxed{MSE \approx 0.336}$

$\boxed{\text{if we don't care about centering, then let } \hat{V} = \alpha U}$

$\Rightarrow P(\alpha) = \mathbb{E}[(V-\alpha U)(V-\alpha U)]$

$= \mathbb{E}[V^2] - 2\alpha R_{uv} + \alpha^2 \cancel{\mathbb{E}[U^2]}^{1}$

$= R_v - 2\alpha R_{uv} + \alpha^2$

$\frac{\partial P(\alpha)}{\partial \alpha} = -2R_{uv} + 2\alpha = 0 \Rightarrow \boxed{\alpha = R_{uv}}$

$\Rightarrow MSE = \mathbb{E}[(V-\alpha U)(V-\alpha U)] = R_v - 2\alpha R_{uv} + \alpha^2$

$= R_v - 2R_{uv}^2 + R_{uv}^2$

$\Rightarrow \boxed{MSE = R_v - R_{uv}^2}$

(un-centered)

$R_v = \mathbb{E}[V^2] = \mathbb{E}[(\sqrt{U^2})^2]$

$R_{uv} = \mathbb{E}[UV] = \mathbb{E}[U\sqrt{U^2}]$

$\boxed{MSE = 0.929}$

Q2) optimal estimator of V from U of form $\alpha + \beta U$ (affine estimator) to minimize mean-square error. provide MSE calc. (assume scalar variables)

~~let $P(\alpha,\beta) = \mathbb{E}((v-\hat{v})(v-\hat{v}))$~~ , $\hat{v} = \alpha + \beta U$ per problem statement

~~$= \mathbb{E}((v-\alpha-\beta U)(v-\alpha-\beta U))$~~

~~$= \mathbb{E}[v^2 - 2\alpha v - 2v\beta U + 2\alpha\beta U + \alpha^2 + \beta^2 U^2]$~~

V is not centered, so define $\tilde{v} \triangleq v - \mu_v$, $\mu_v = \mathbb{E}[V]$

now, $\hat{\tilde{v}} = \alpha + \beta U$

$$P(\alpha,\beta) = \mathbb{E}((\tilde{v}-\hat{\tilde{v}})(\tilde{v}-\hat{\tilde{v}})) = \mathbb{E}[(\tilde{v}-\alpha-\beta U)(\tilde{v}-\alpha-\beta U)]$$

$$= \mathbb{E}[\tilde{v}^2 - 2\alpha\tilde{v} - 2\tilde{v}\beta U + 2\alpha\beta U + \alpha^2 + \beta^2 U^2]$$

$$= R_{\tilde{v}} - 2\alpha\mu_{\tilde{v}} - 2\beta R_{\tilde{v}u} + 2\alpha\beta\mu_u^{\,0} + \alpha^2 + \beta^2 R_u^{\,1}$$

$$= R_{\tilde{v}} - 2\alpha\mu_{\tilde{v}} - 2\beta R_{\tilde{v}u} + \alpha^2 + \beta^2$$

$$\frac{\partial P(\alpha,\beta)}{\partial\alpha} = 0 - 2\mu_{\tilde{v}} - 0 + 2\alpha = 0 \Rightarrow \boxed{\alpha = \mu_{\tilde{v}} = \mathbb{E}(v-\mu_v) = \mu_v - \mu_v = 0}$$

$$\frac{\partial P(\alpha,\beta)}{\partial\beta} = 0 - 0 - 2R_{\tilde{v}u} + 0 + 2\beta = 0 \Rightarrow \boxed{\beta = R_{\tilde{v}u}}$$

$$R_{\tilde{v}u} = \mathbb{E}[\tilde{v}u] = \mathbb{E}[(v-\mu_v)u] = R_{vu}$$

$$\boxed{\Rightarrow \beta = R_{vu}}$$

$MSE = \mathbb{E}[(v-\hat{v})(v-\hat{v})]$, where $\hat{\tilde{v}} = \alpha + \beta u \Rightarrow \hat{v} - \mu_v = \alpha + \beta u$

$\Rightarrow \hat{v} = \alpha + \beta u + \mu_v$

$= R_{vu} U + \mu_v$

$$= \mathbb{E}[(v - R_{vu}U - \mu_v)(v - R_{vu}U - \mu_v)]$$

$\Rightarrow \boxed{MSE = R_v - R_{vu}^2 - \mu_v^2}$, per [Q1]

$R_{vu} = \mathbb{E}[vu] = \mathbb{E}[\sqrt{u^2}\, u]$

$R_v = \mathbb{E}[|u|^2] = \mathbb{E}[u^2] = 1$

$\mu_v = \mathbb{E}(v) = \mathbb{E}[\sqrt{u^2}]$

→ see code for estimate of MSE

$\boxed{MSE \approx 0.336}$

Q3 opt. est. of $V|U$, form $\alpha + \beta U + \gamma U^2$ → quadratic estimator for min. MSE.

→ provide MSE estimate.

V not centered, so define $\tilde{v} = v - \mu_v$, $\mu_v = E(v) = \sum_{i=1}^{n} v_i / n$

now, $\hat{\tilde{v}} = \alpha + \beta u + \gamma u^2$ (quadratic estimator for v)

then, $P(\alpha, \beta, \gamma) = E[(\tilde{v} - \hat{\tilde{v}})(\tilde{v} - \hat{\tilde{v}})]$

$= E[(\tilde{v} - \alpha - \beta u - \gamma u^2)(\tilde{v} - \alpha - \beta u - \gamma u^2)]$ → MATLAB

$= E[\alpha^2 + 2\alpha\beta u + 2\alpha\gamma u^2 - 2\alpha\tilde{v} + \beta^2 u^2 + 2\beta\gamma u^3 - 2\beta u\tilde{v} + \gamma^2 u^4 - 2\gamma u^2 \tilde{v} + \tilde{v}^2]$

$= \alpha^2 + 2\alpha\beta(0) + 2\alpha\gamma(1) - 2\alpha\mu_{\tilde{v}} + \beta^2 + \cancel{2\beta\gamma(0)} - 2\beta R_{\tilde{v}u} + \gamma^2 3 - 2\gamma E[u^2\tilde{v}] + \tilde{v}^2]$

$= \alpha^2 + 2\alpha\gamma - 2\alpha\mu_{\tilde{v}} + \beta^2 - 2\beta R_{\tilde{v}u} + 3\gamma^2 - 2\gamma E(u^2\tilde{v}) + E(\tilde{v}^2)$

$E(u^2\tilde{v}) = E[u^2(v - \mu_v)] = E[u^2 v] - \mu_v$

$\frac{\partial P(\alpha, \beta, \gamma)}{\partial \alpha} = 2\alpha + 2\gamma - 2\mu_{\tilde{v}}^{\,0} = 0 \Rightarrow \alpha = \mu_{\tilde{v}}^{\,0} - \gamma$

$\frac{\partial P}{\partial \beta} = 2\beta - 2R_{\tilde{v}u} = 0 \Rightarrow \boxed{\beta = R_{\tilde{v}u} = R_{vu}}$

$\frac{\partial P}{\partial \gamma} = 2\alpha + 6\gamma - 2E[u^2\tilde{v}] = 0 \Rightarrow \alpha + 3\gamma - E[u^2\tilde{v}] = 0$, let $E[u^2\tilde{v}] = \emptyset$

$\Rightarrow \mu_{\tilde{v}} - \gamma + 3\gamma - \emptyset = 0$ $\quad = E[u^2\sqrt{u^2}] - \mu_v$

$E[\tilde{v}] = E[v - \mu_v] = 0$

$\Rightarrow 2\gamma = \emptyset - \mu_{\tilde{v}}^{\,0}$

$\Rightarrow \boxed{\gamma = (\emptyset - \mu_{\tilde{v}}^{\,0})/2}$

$\gamma = \frac{(E[u^2 v] - \mu_v)}{2}$ ✓

$\Rightarrow \alpha = \mu_{\tilde{v}}^{\,0} - \frac{(\emptyset - \mu_{\tilde{v}})}{2}$

$\emptyset = E[u^2\tilde{v}] = E[u^2(v - \mu_v)]$

$= E[u^2 v] - \mu_v$

$\boxed{\alpha = (3\mu_{\tilde{v}}^{\,0} - \emptyset)/2}$

$\alpha = -\frac{\emptyset}{2}$

$\gamma = \frac{\emptyset}{2}$ $\quad$ MSE ⟶

$\underline{MSE}$ $= \mathbb{E}[(v-\hat{v})(v-\hat{v})]$, $\hat{\tilde{v}} = \hat{v} - \mu_v = \alpha + \beta u + \gamma u^2$

$\Rightarrow \hat{v} = \alpha + \beta u + \gamma u^2 + \mu_v$

$\Rightarrow MSE = \mathbb{E}[(v-(\alpha+\beta u+\gamma u^2+\mu_v))^2] = \mathbb{E}[v^2 - 2v\hat{v} + \hat{v}^2]$

$= \mathbb{E}[v^2] - 2\mathbb{E}[v(\alpha+\beta u+\gamma u^2+\mu_v)] + \mathbb{E}[(\alpha+\beta u+\gamma u^2+\mu_v)^2]$

$= R_v - 2(\alpha\mu_v + \cancel{0+\gamma\,2\mu_v^2} + \beta R_{uv} + \gamma\mathbb{E}(u^2v) + \mu_v^2) + \mathbb{E}[(\alpha+\beta u+\gamma u^2+\mu_v)^2]$

$= R_v - 2(\alpha\mu_v + \beta R_{uv} + \gamma\mathbb{E}(u^2v) + \mu_v^2)$
$+ \underbrace{\mathbb{E}(\alpha^2 + 2\alpha\beta u + 2\alpha\gamma u^2 + 2\alpha\mu_v + \beta^2u^2 + 2\beta\gamma u^3 + 2\beta\mu_v u + \gamma^2u^4 + 2\gamma\mu_v u^2 + \mu_v^2)}$

$= \mathbb{E}\cancel{\alpha}$
$= \alpha^2 + 0 + 2\alpha\gamma + 2\alpha\mu_v + \beta^2 + 0$
$+ 0 + 3\gamma^2 + 2\gamma\mu_v + \mu_v^2$

principle of orthogonality?

$\mathbb{E}[(\tilde{v}-\hat{\tilde{v}})(\tilde{v}-\hat{\tilde{v}})]$

$= \mathbb{E}[(\tilde{v}-\hat{\tilde{v}})\tilde{v} - (\tilde{v}-\hat{\tilde{v}})\hat{\tilde{v}}]$ due to orthogonality

$= \mathbb{E}[(\tilde{v}-\hat{\tilde{v}})\tilde{v}] - 0$

$= \mathbb{E}[\tilde{v}^2 - (\alpha+\beta u+\gamma u^2)\tilde{v}]$

$= R_{\tilde{v}} - \alpha\mu_{\tilde{v}} - \beta R_{\tilde{v}u} - \gamma\mathbb{E}[u^2\tilde{v}]$

$\mathbb{E}(u^3) = 0$

$\mathbb{E}(u^4) = 3\sigma^2 = 3$

$\Rightarrow$

$MSE = R_v - 2(\cancel{\alpha\mu_v} + \beta R_{uv} + \gamma\mathbb{E}[u^2v] + \mu_v^2) + \alpha^2 + 2\alpha\gamma + \cancel{2\alpha\mu_v} + \beta^2 + 3\gamma^2 + 2\gamma\mu_v + \mu_v^2$

$= R_v - 2\beta R_{uv} - 2\gamma\mathbb{E}[u^2v] - \cancel{2\mu_v^2} + \alpha^2 + 2\alpha\gamma + \beta^2 + 3\gamma^2 + 2\gamma\mu_v + \mu_v^2$

$\alpha =$

What if we let $\alpha = \mu_v$ from beginning?

Simpler method $\longrightarrow$

let $\hat{v} = \alpha + \beta u + \gamma u^2$ (form of estimate)

since v is not centered, then $\tilde{v} \triangleq v - \mu_v$ s.t. $\hat{v} = \hat{\tilde{v}} + \mu_v$

$$\Rightarrow \hat{v} = \alpha + \beta u + \gamma u^2 = \hat{\tilde{v}} + \mu_v \quad \boxed{\text{let } \alpha = \mu_v} \text{ s.t. } \boxed{\hat{\tilde{v}} = \beta u + \gamma u^2}$$

then, $\rho(\alpha,\beta,\gamma) = \mathbb{E}[(\tilde{v} - \hat{\tilde{v}})(\tilde{v} - \hat{\tilde{v}})] = \mathbb{E}[(\tilde{v} - \beta u - \gamma u^2)(\tilde{v} - \beta u - \gamma u^2)]$

$$= \mathbb{E}[\tilde{v}^2 + \beta^2 u^2 + \gamma^2 u^4 - \beta u \tilde{v} - \beta u \tilde{v} - 2\gamma u^2 \tilde{v} + 2\beta u^3 \gamma]$$

$$= \mathbb{E}[\tilde{v}^2] + \beta^2(1) + \gamma^2(3) - 2\beta R_{u\tilde{v}} - 2\gamma \mathbb{E}(u^2 \tilde{v}) + 2\beta\gamma \cancel{\mathbb{E}(u^3)}^{0}$$

$$= R_{\tilde{v}} + \beta^2 + 3\gamma^2 - 2\beta R_{uv} - 2\gamma \mathbb{E}(u^2 \tilde{v})$$

$$\frac{\partial \rho}{\partial \alpha} = 0$$

$$\frac{\partial \rho}{\partial \beta} = 2\beta - 2R_{uv} = 0 \Rightarrow \boxed{\beta = R_{uv}}$$

$$\frac{\partial \rho}{\partial \gamma} = 6\gamma - 2\mathbb{E}[u^2 \tilde{v}] = 0 \Rightarrow \gamma = \frac{1}{3}\mathbb{E}[u^2 \tilde{v}] = \frac{1}{3}\mathbb{E}[u^2(v - \mu_v)] = \frac{1}{3}(\mathbb{E}(u^2 v) - \mu_v)$$

$$MSE = \mathbb{E}[\cancel{(\tilde{v} - \hat{\tilde{v}})(\tilde{v} - \hat{\tilde{v}})}] = \mathbb{E}[(v - (\hat{\tilde{v}} + \overset{\alpha}{\mu_v}))(v - (\hat{\tilde{v}} + \overset{\alpha}{\mu_v}))]$$

$$= \mathbb{E}[(v - \hat{v})(v - \hat{v})] = \mathbb{E}[v^2] - 2\mathbb{E}[v\hat{v}] + \mathbb{E}[\hat{v}^2]$$

note: $\mathbb{E}[u^3] = 0$

$\mathbb{E}[u^4] = 3$

where

$$\mathbb{E}[v\hat{v}] = \mathbb{E}[v(\beta u + \gamma u^2 + \alpha)] = \beta R_{uv} + \gamma \mathbb{E}[u^2 v] + \mu_v^2$$

$$\mathbb{E}[\hat{v}^2] = \mathbb{E}[(\mu_v + \beta u + \gamma u^2)^2] = \mathbb{E}[\mu_v^2 + \beta^2 u^2 + \gamma^2 u^4 + \cancel{2\beta\mu_v u}^{0} + 2\mu_v \gamma u^2 + \cancel{2\beta\gamma u^3}^{0}]$$

$$= \mu_v^2 + \beta^2 + 3\gamma^2 + 2\mu_v \gamma$$

thus,

$$\boxed{\begin{aligned} &MSE = R_v - 2(\beta R_{uv} + \gamma \mathbb{E}(u^2 v) + \mu_v^2) + (\mu_v^2 + \beta^2 + 3\gamma^2 + 2\mu_v \gamma) \\ &\beta = R_{uv} = \mathbb{E}[u|u|] = \mathbb{E}[u\sqrt{u^2}] \qquad \mathbb{E}[u^2 v] = \mathbb{E}[u^2\sqrt{u^2}] \\ &\gamma = \tfrac{1}{3}(\mathbb{E}[u^2\sqrt{u^2}] - \mathbb{E}[\sqrt{u^2}]) \\ &R_v = \mathbb{E}[(\sqrt{u^2})^2] \end{aligned}}$$

estimate $\rightarrow$

to numerically estimate, generate $n = 1e3$ or $1e4$ points per $U \sim N(0,1)$ and then actually calculate the MSE for each problem!

MSE summary:

$\alpha = \mu_v$

$\beta = R_{uv} = R_{vu}$

$\gamma = \frac{1}{3}\left(\mathbb{E}[u^2 v] - \mu_v\right) \rightarrow$ let $\emptyset = \mathbb{E}[u^2 v]$

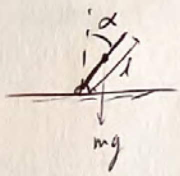
$\Rightarrow \gamma = \frac{1}{3}(\emptyset - \mu_v)$

$$MSE = R_v - 2\left(\beta R_{vu} + \gamma \emptyset + \mu_v^2\right) + \left(\mu_v^2 + \beta^2 + 3\gamma^2 + 2\mu_v \gamma\right)$$

$\rightarrow$ $\boxed{\text{see python for code to estimate}}$

$\rightarrow$ $\boxed{MSE \approx 0.1815}$

Problem 2



$l = 1$
$m = 3.5\,kg$
$g = 9.8\ m/s^2$

EOM:

$$\frac{d^2\alpha}{dt^2} = -\frac{g}{l}\sin\alpha + w(t)$$

define state as: $\mathbf{x} \triangleq \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \triangleq \begin{bmatrix} \alpha \\ \frac{d\alpha}{dt} \end{bmatrix}$

Q1] State Space model:

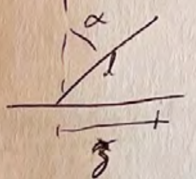
$$\dot{x} = \begin{bmatrix} \frac{d\alpha}{dt} \\ \frac{d^2\alpha}{dt} \end{bmatrix} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ -\frac{g}{l}\sin\alpha + w(t) \end{bmatrix} = \begin{bmatrix} x_2 \\ -\frac{g}{l}\sin(x_1) \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} w(t)$$

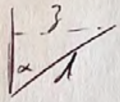
$\Rightarrow$ form: $\dot{x} = \begin{bmatrix} f_1(x_2) \\ f_2(x_1) \end{bmatrix} + G\,w(t) \Rightarrow$

$$\boxed{\begin{aligned} f_1(x_2) &= x_2 \\ f_2(x_1) &= -\frac{g}{l}\sin(x_1) \\ G &= \begin{bmatrix} 0 \\ 1 \end{bmatrix} \end{aligned}}$$

Q2] Sensor model:

show $y = l\sin x_1 + v(t)$ i.e. tracks horizontal position + additive noise





$\cancel{l}\sin\alpha = \frac{z}{\cancel{l}} \Rightarrow z = l\sin\alpha = l\sin x_1$

thus, $\boxed{y = l\sin x_1 + v(t)}$

Q3] Discretize:

Discretization of step Δt s.t. $\frac{dx}{dt} = \frac{x(t+\Delta t) - x(t)}{\Delta t}$

let $x_k \triangleq x(k\Delta t)$ so that $\left.\frac{dx}{dt}\right|_{\Delta t = k\Delta t} \approx \frac{x_{k+1} - x_k}{\Delta t}$

Show the discretized model takes the following form:

$$\begin{bmatrix} x_{1,k+1} \\ x_{2,k+1} \end{bmatrix} = \begin{bmatrix} x_{1,k} + x_{2,k}\Delta t \\ x_{2,k} - g\Delta t \sin(x_{1,k}) \end{bmatrix} + q_{k+1}$$

$$y_{k+1} = \sin x_{1,k+1} + v_{k+1}$$

note: (instead of a_{k+1}, use v_{k+1} for consistency)

process:

$$\dot{x}_1 = x_2 \Rightarrow \frac{x_{1,k+1} - x_{1,k}}{\Delta t} = x_{2,k} \Rightarrow x_{1,k+1} = x_{1,k} + \Delta t\, x_{2,k}$$

$$\dot{x}_2 = \cancel{x_2} - g/l\, \sin(x_1) + w(t) \qquad (l = 1)$$

$$w(\Delta t k) = w_k$$

$$\Rightarrow \frac{x_{2,k+1} - x_{2,k}}{\Delta t} = -g \sin(x_{1,k}) + w(t)$$

$$\Rightarrow x_{2,k+1} = x_{2,k} - g\Delta t \sin(x_{1,k}) + \Delta t\, w_{k}$$

then,

$$\boxed{\begin{bmatrix} x_{1,k+1} \\ x_{2,k+1} \end{bmatrix} = \begin{bmatrix} x_{1,k} + x_{2,k}\Delta t \\ x_{2,k} - g\sin(x_{1,k})\Delta t \end{bmatrix} + q_{k}, \quad q_{k} = \begin{bmatrix} 0 \\ \Delta t\, w_{k} \end{bmatrix}}$$

$$q_k = \begin{bmatrix} 0 \\ \Delta t\, w_k \end{bmatrix}$$

measurement:

$$y(t) = l \sin(x_1) + v(t), \quad l = 1$$

$$\Rightarrow y(\Delta t k) = \sin(x_1(\Delta t k)) + v(\Delta t k) \Rightarrow y_k = \sin(x_{1,k}) + v_k$$

same as:

$$\boxed{y_{k+1} = \sin(x_{1,k+1}) + v_{k+1}}$$

note: if $w(t)$ is white Gaussian, then q_{k+1} is white Gaussian w/ covariance matrix:

$$Q \triangleq \sigma_p^2 \begin{bmatrix} \frac{\Delta t^3}{3} & \frac{\Delta t^2}{2} \\ \frac{\Delta t^2}{2} & \Delta t \end{bmatrix}$$

note 2: v_{k+1} is assumed white Gaussian w/ variance σ_m^2

Q4] ~~linearize~~ for EKF

Form of model: $x_{k+1} = f(x_k) + q_k$, $y_{k+1} = h(x_{k+1}) + v_{k+1}$

Show that the linearization of functions f and h around a state

$x^* \triangleq \begin{bmatrix} x_1^* \\ x_2^* \end{bmatrix}$ take the form:

$$f(x) \approx f(x^*) + \begin{bmatrix} 1 & \Delta t \\ -g\cos(x_1^*)\Delta t & 1 \end{bmatrix}(x - x^*)$$

$$h(x) \approx h(x^*) + [\cos(x_1^*) \quad 0](x - x^*)$$

note: $f(x) \approx f(x^*) + F(x^*)(x - x^*)$ where $F(x^*) \triangleq \left. \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & & \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} \right|_{x = x^*}$

$\cancel{F(x^*)} = f(x) = \begin{bmatrix} f_1(x) \\ f_2(x \end{bmatrix}$, $\cancel{\frac{\partial f_1}{\partial x_1}} =$ $f_1(x_k) = x_{1,k} + x_{2,k}\Delta t$

$f_2(x_k) = x_{2,k} - g\sin(x_{1,k})\Delta t$

$$\left.\begin{aligned} \frac{\partial f_1}{\partial x_1} &= 1 \\ \frac{\partial f_1}{\partial x_2} &= \Delta t \\ \frac{\partial f_2}{\partial x_1} &= -g\Delta t\cos(x_{1,k}) \\ \frac{\partial f_2}{\partial x_2} &= 1 \end{aligned}\right\} \Rightarrow F(x^*) = \begin{bmatrix} 1 & \Delta t \\ -g\Delta t\cos(x_{1,k}^*) & 1 \end{bmatrix}$$

thus, ~~the~~ 1st order approximation of $f(x_k)$ is:

$$\boxed{f(x_k) \approx f(x_k^*) + \begin{bmatrix} 1 & \Delta t \\ -g\Delta t\cos(x_{1,k}^*) & 1 \end{bmatrix}(x_k - x_k^*)}$$

note: $f(x_k^*) = \begin{bmatrix} x_{1,k}^* + x_{2,k}^*\Delta t \\ x_{2,k}^* - g\sin(x_{1,k}^*)\Delta t \end{bmatrix}$

for the measurement equation;

$$\cancel{h(x)}\ h(x_k) \approx h(x_k^*) + H(x_k^*)(x_k - x_k^*) \quad , \quad h(x_k) = \sin(x_{1,k})$$

$$h(x_k^*) = \sin(x_{1,k}^*)$$

$$H(x_k^*) = \begin{bmatrix} \frac{\partial h}{\partial x_1} & \frac{\partial h}{\partial x_2} \end{bmatrix} \quad , \quad \frac{\partial h}{\partial x_1} = \cos(x_{1,k})$$

$$\Rightarrow H(x_k^*) = \begin{bmatrix} \cos(x_{1,k}^*) & 0 \end{bmatrix}$$

$$\frac{\partial h}{\partial x_2} = 0$$

then,

$$\boxed{h(x_k) \approx \sin(x_{1,k}^*) + \begin{bmatrix} \cos(x_{1,k}^*) & 0 \end{bmatrix} (x - x_k^*)}$$

Implement the EKF!

parameters: $\sigma_p = 0.1$, $\sigma_m = 0.3$, $\Delta t = 20\text{ms}$

ground truth.npy → true trajectory sampled at 1ms

measurements.npy → noisy measurements, sampled at 20ms

Q5 EKF equations of the code: (from class lecture, Lec 17, pg 6)

$$\hat{x}_{k+1|k} = f_k(\hat{x}_{k|k})$$

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_{f,k}\left(y_k - h_k(\hat{x}_{k|k-1})\right)$$

$$K_{f,k} = P_{k|k-1} H(\hat{x}_{k|k-1})^T \left(H(\hat{x}_{k|k-1}) P_{k|k-1} H(\hat{x}_{k|k-1})^T + R_k\right)^{-1}$$

$$P_{k|k} = \left(I - K_{f,k} H(\hat{x}_{k|k-1})\right) P_{k|k-1}$$

$$P_{k+1|k} = F(\hat{x}_{k|k}) P_{k|k} F(\hat{x}_{k|k})^T + Q_k$$

→

$R_k = \mathbb{E}[v_k^2] = \sigma_m^2$ (variance of v_k)

$$Q_k = \sigma_p^2 \begin{bmatrix} \Delta t^3/3 & \Delta t^2/2 \\ \Delta t^2/2 & \Delta t \end{bmatrix}$$

$\sigma_m = 0.3$

$\sigma_p = 0.1$

$\Delta t = 20\,ms$

$f_k(x_k) = f(x_k)$

$h_k(x_k) = h(x_k)$

$$f_k(\hat{x}_{k|k}) = \begin{bmatrix} \hat{x}_{1,k|k} + \hat{x}_{2,k|k}\Delta t \\ \hat{x}_{2,k|k} - g\Delta t \sin(\hat{x}_{1,k|k}) \end{bmatrix}$$

$$F(\hat{x}_{k|k}) = \begin{bmatrix} 1 & \Delta t \\ -g\Delta t \sin(\hat{x}_{1,k|k}) & 1 \end{bmatrix}$$

$$h_k(\hat{x}_{k|k-1}) = \sin(\hat{x}_{1,k|k-1})$$

$$H(\hat{x}_{k|k-1}) = \begin{bmatrix} \cos(\hat{x}_{1,k|k-1}) & 0 \end{bmatrix}$$

→ See python code for implementation

RMS error: $\sqrt{\frac{\Sigma(\hat{x}_{1,k|k} - x)^2}{n}} \approx$ ~~4.03~~ 0.180

$\sqrt{\frac{\Sigma(y-x)^2}{n}} = 0.311$

Particle Filter →

Particle Filter:

from 2-Q3), we have the discretized non-linear model:

$$\begin{bmatrix} x_{1,k+1} \\ x_{2,k+1} \end{bmatrix} = \begin{bmatrix} x_{1,k} + x_{2,k}\Delta t \\ x_{2,k} - g\Delta t \sin(x_{1,k}) \end{bmatrix} + q_k \quad \text{where } q_k = \begin{bmatrix} 0 \\ \Delta t\, w_k \end{bmatrix}$$

$$y_{k+1} = \sin(x_{1,k+1}) + v_{k+1}$$

$\{v_k\}$ and $\{w_k\}$ are white, Gaussian, $R_v = \sigma_m^2$, $\sigma_m = 0.3$

$\{q_k\}$ is white, Gaussian w/ $R_q = Q_k \triangleq \sigma_p^2 \begin{bmatrix} \Delta t^3/3 & \Delta t^2/2 \\ \Delta t^2/2 & \Delta t \end{bmatrix}$, $\sigma_p = 0.1$

with some a-priori knowledge of the statistics of the system, set

$x_0^{(i)} \sim N(y_0, 0.5)$ for $n = 200$ particles, $i = 1 \ldots n$

$w^{(i)} = 1/n$

then we have step 1) of a PF: draw n samples from the prior and set weights to $1/n$

now

2) For each $k = 1 \ldots T$

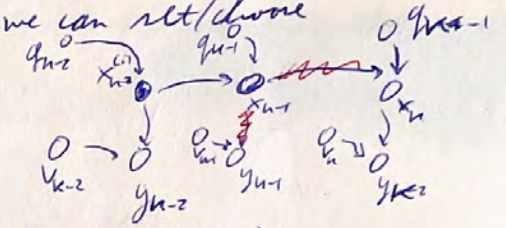
a) draw samples $x_k^{(i)}$ from importance distribution

$$x_k^{(i)} \sim \pi(x_k \mid x_{0:k-1}^{(i)}, y_{0:k}) \qquad i = 1 \ldots n$$

b) compute new weights

$$w_k^{(i)} \propto w_{k-1}^{(i)} \frac{p(y_k \mid x_k^{(i)})\, p(x_k^{(i)} \mid x_{k-1}^{(i)})}{\pi(x_k^{(i)} \mid x_{0:k-1}^{(i)}, y_{0:k})}$$ and normalize

So, b/c we have the state-space model, we can set/choose



$$\pi(x_k^{(i)} | x_{0:k-1}^{(i)}\ y_{0:k}) \triangleq p(x_k^{(i)} | x_{k-1}^{(i)})$$

(i.e. only push the current particle through the process update)

thus, $p(x_k^{(i)} | x_{k-1}^{(i)}) = p\left(\begin{bmatrix} x_{1,k-1} + x_{2,k-1}\Delta t \\ x_{2,k-1} - g\Delta t \sin(x_{1,k-1}) \end{bmatrix} + q_{k-1} \;\middle|\; x_{k-1}^{(i)} \right)$

$$= \underbrace{\begin{bmatrix} x_{1,k-1}^{(i)} + x_{2,k-1}^{(i)}\Delta t \\ x_{2,k-1}^{(i)} - g\Delta t \sin(x_{1,k-1}^{(i)}) \end{bmatrix}}_{(PF-2)} + \underbrace{p(q_{k-1} | x_{k-1}^{(i)})}$$

$= p(q_{k-1})$ (noise is independent of past & current states)

$= 0$ ← white noise Gaussian ~~when realized~~

$$\boxed{\Rightarrow p(x_k^{(i)} | x_{k-1}^{(i)}) = \begin{bmatrix} x_{1,k-1}^{(i)} + x_{2,k-1}^{(i)}\Delta t \\ x_{2,k-1}^{(i)} - g\Delta t \sin(x_{1,k-1}^{(i)}) \end{bmatrix} + p(q_{k-1})} \quad (PF-1)$$

therefore, when we sample from the importance distribution, given $x_{k-1}^{(i)}$, the result is (PF-2) plus the realization of $\{q_k\}$

Now compute expression for the update of the weights:

$$w_k^{(i)} \propto w_{k-1}^{(i)} \frac{p(y_i | x_k^{(i)})\, \cancel{p(x_k^{(i)} | x_{k-1}^{(i)})}}{\underbrace{\cancel{\pi(x_k^{(i)} | x_{0:k-1}^{(i)}\ y_{0:k})}}_{\triangleq \cancel{p(x_k^{(i)} | x_{k-1}^{(i)})}}} = w_{k-1}^{(i)}\, p(y_i | x_k^{(i)})$$

→

to solve for $p(y_k | x_{k-1}^{(i)})$, we note that y_k is Gaussian due to the Gaussian noise $\{v_k\}$. thus, $v_k \sim N(0, \sigma_m^2)$

$$\cancel{y_k \sim N(\mathbb{E}[y_k]}, \quad y_k | x_k^{(i)} \sim N(\mathbb{E}[y_k | x_k^{(i)}], K_v)$$

$$\boxed{\mathbb{E}[y_k | x_k^{(i)}] = E[\sin(x_k) + v_k | x_k^{(i)}] = \sin(x_k^{(i)}) \triangleq \mu_y} \quad (PF-3)$$

$$K_y = \mathbb{E}[(y-\mu_y)(y-\mu_y) | x_k^{(i)}] = \mathbb{E}[y^2 - 2\mu_y y + \mu_y^2 | x_k^{(i)}]$$

$$= \mathbb{E}[(\sin(x_k) + v_k)^2] - \cancel{2}\mu_y^2 + \cancel{\mu_y^2}$$

$$= \mathbb{E}[\sin^2(x_k) + 2\sin(x_k) v_k + v_k^2 | x_k^{(i)}] - \mu_y^2$$

$$= \cancel{\sin^2(x_k^{(i)})} + 2\sin(x_k^{(i)}) \underbrace{\mathbb{E}[v_k | x_k^{(i)}]}_{\to 0} + R_v - \cancel{\sin^2(x_k^{(i)})}$$

$$\Rightarrow \boxed{K_y = R_v = \sigma_m^2} \quad (PF-4)$$

thus, $$\boxed{y_k | x_k^{(i)} \sim N(\sin(x_k^{(i)}), \sigma_m^2)} \quad (PF-5)$$

and $$\boxed{p_{y_k}(y_k | x_k^{(i)}) = \frac{1}{\sigma_m \sqrt{2\pi}} \exp\left(-\frac{1}{2} \frac{(y_k - \sin(x_k^{(i)}))^2}{\sigma_m^2}\right)} \quad (PF-6)$$

such that

$$w_k^{(i)} \propto w_{k-1}^{(i)} p(y_k | x_k^{(i)})$$

$$\Rightarrow \boxed{w_k^{(i)} \propto w_{k-1}^{(i)} \left[\frac{1}{\sigma_m \sqrt{2\pi}} \exp\left(-\frac{1}{2} \frac{(y_k - \sin(x_k^{(i)}))^2}{\sigma_m^2}\right)\right]} \quad (PF-7)$$

then, $w_k^{(i)} = \dfrac{w_k^{(i)}}{\sum_{i=1}^{n} w_k^{(i)}}$ to normalize

PF part 3) re-sampling:

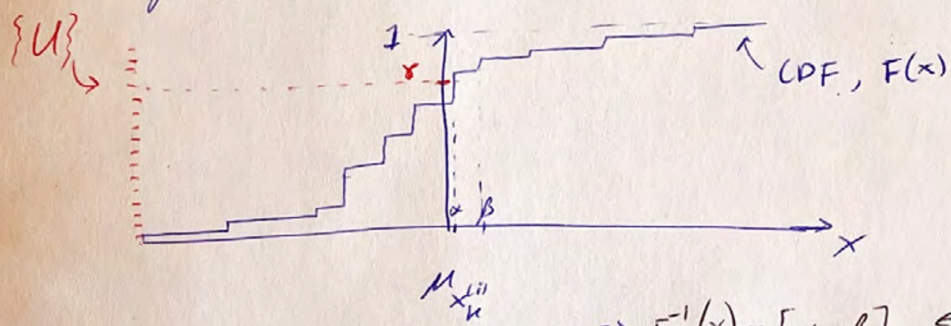
Re-sample if $\eta_{eff} < 20$ (i.e. 10%)

let $\eta_{eff} \approx \dfrac{1}{\sum_{j=1}^{n} (w_k^{(j)})^2}$

per HW#5 of ECE 6555, we can re-sample according $F^{-1}(u)$ where U is a uniform distribution of particles [0,1] and

$F = \sum_{i=1}^{n} w_k^{(i)} \delta(x - x_k^{(i)})$ where $x_k^{(i)}$ are sorted (~~and~~ low → high.

basically this would look something like:



$\Rightarrow F^{-1}(\gamma) = [\alpha, \beta]$ ← in code, I will sample from this range.

See python code for implementation

note: did not know how to deal w/ 2-D state model... code will not function

Table of Contents

ECE6555 HW5

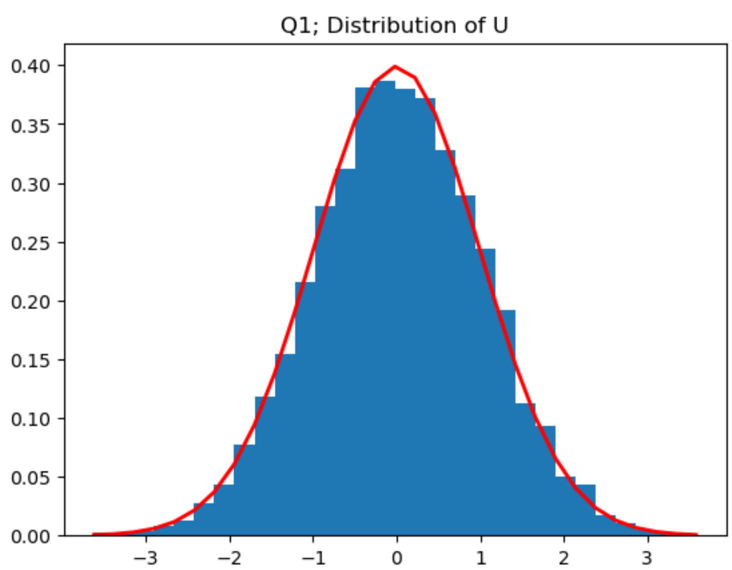
Author: Teo Wilkening

Due Date: 2022-12-16

1 [1-Q1] Optimal estimator of V from U of the form αU

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

In [2]:
```python
u_mu, u_sigma = 0, 1
n = 10000 # number of samples
u = np.random.normal(u_mu, u_sigma,n)
count, bins, ignored = plt.hist(u, 30, density=True)
plt.plot(bins, 1/(u_sigma * np.sqrt(2 * np.pi)) *
               np.exp( - (bins - u_mu)**2 / (2 * u_sigma**2) ),
         linewidth=2, color='r')
plt.title('Q1; Distribution of U')
plt.show()
```



1.1 MSE numerical estimate

In [3]:
```
v = np.sqrt(u**2)
Rv  = np.sum(v**2)/n
Rvu = np.sum(u*v)/n
v_mu = np.sum(v)/n
MSE_linear = Rv - Rvu**2 - v_mu**2
print(f"""The Mean Square error of the linear estimate (for v centered) is: {MSE_linear}""")

MSE_uncentered = Rv - Rvu**2
print(f"""The Mean Square error of the linear estimate (for v un-centered) is: {MSE_uncentered}""")

```

```
The Mean Square error of the linear estimate (for v centered) is: 0.362708823099285
The Mean Square error of the linear estimate (for v un-centered) is: 1.0130574309689977
```

2 [1-Q2] Optimal estimator of V from U of the form $\alpha + \beta U$

In [4]:
```
MSE_affine = Rv - Rvu**2 - v_mu**2
print(f"""The Mean Square error of the affine estimate (for v centered) is: {MSE_affine}""")
```

```
The Mean Square error of the affine estimate (for v centered) is: 0.362708823099285
```

3 [1-Q3] Optimal estimator of V from U of the form $\alpha + \beta U + \gamma U^2$

In [5]:
```
alpha = v_mu
beta = Rvu
phi = np.sum((u**2)*v)/n
gamma = 1/3*(phi - v_mu)
MSE_quadratic = Rv - 2*(beta*Rvu + gamma*phi + v_mu**2) + (v_mu**2 + beta**2 + 3*gamma**2 + 2*v_mu*gamma)
print(f"""The Mean Square error of the quadratic estimate (for v centered) is: {MSE_quadratic}""")
```

```
The Mean Square error of the quadratic estimate (for v centered) is: 0.1423163012004771
```

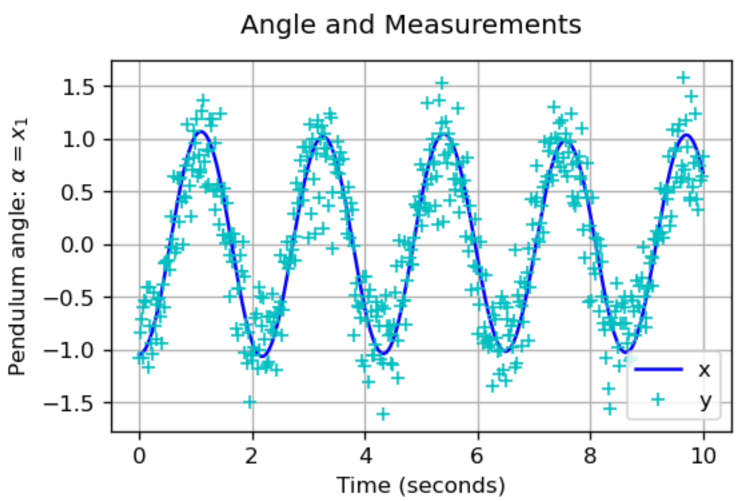
4 [2-Q6] EKF implementation

(will be borrowing my code from HW #4)

4.1 Plot the measured and truth data for visualization

In [6]:

```
import numpy as np
from scipy import signal

tscale, x = np.load("groundtruth.npy") # ground truth at 1ms
tscale_measurement, y = np.load("measurements.npy") # sampled at 20ms
tscale_measurement2, y2 = np.load("measurements2.npy") # sampled at 2ms

fig, ax = plt.subplots(figsize=(5,3), dpi=120)

ax.plot(tscale,x,'b')
ax.grid(True)
ax.plot(tscale_measurement,y,'c+')
ax.legend(['x','y'])
ax.set_ylabel(r'Pendulum angle: $\alpha = x_1$')
ax.set_xlabel(r'Time (seconds)')
fig.suptitle('Angle and Measurements')

plt.show()
```



4.2 Initialize the necessary parameters/variables

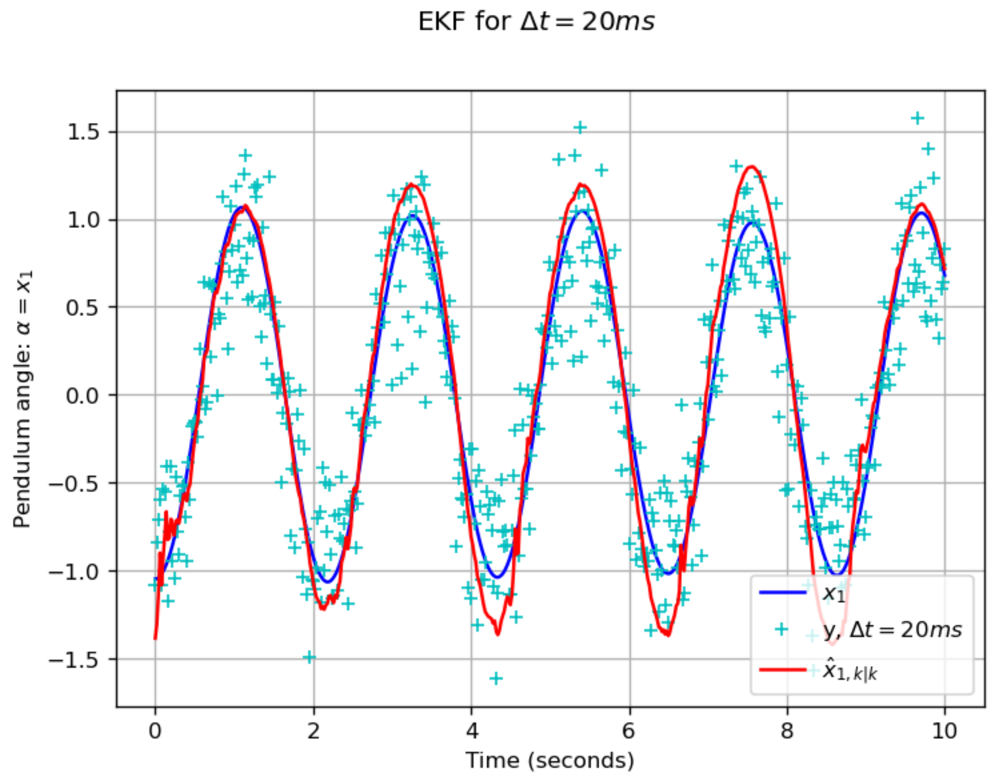
In [7]:

```
# initialize all of the variables that we're going to need
Delta = 0.020 # 20 ms
sigma_m = 0.3
sigma_p = 0.1
Qk = (sigma_p**2)*np.matrix([[Delta**3/3,Delta**2/2],
                             [Delta**2/2,Delta]])
Rk = np.matrix([sigma_m**2])
g = 9.8 # m/s^2

def f(xk,dt):
    fk_x = np.matrix([[xk[0][0] + xk[1][0]*dt],
                      [xk[1][0] - g*dt*np.sin(xk[0][0])]])
    return fk_x

def F(xhat,dt):
    F_x = np.matrix([[1                         , dt],
                     [-g*dt*np.sin(xhat[0][0]), 1]])
    return F_x

def h(xk,dt=None):
    return np.sin(xk[0][0])

def H(xhat,dt=None):
    return np.matrix([np.cos(xhat[0][0]), 0])

# # display Qk
# print(f"""Qk = {Qk} \n""")
# display(Rk)

# # test function f:
# xhat_kk = np.array([[1],[2]])
# print("f")
# display(f(xhat_kk,Delta))

# # test function F:
# print("xhat_kk")
# display(xhat_kk[0][0])
# print("F:")
# display(F(xhat_kk,Delta))

# # test function h
# xhat_km1 = np.array([[1],[2]])
# print("h:")
# display(h(xhat_km1))

# # test function K
# print("H:")
# display(H(xhat_km1))
```

4.3 Implement the EKF for measurements.npy

In [8]:

```python
# the initial guesses of x and P
xhat_init = np.array([[y[0]],[0]]) # set alpha = y[0] and d(alpha)/dt = 0
P_init = np.identity(2)
NumSteps = len(y)

# initializing the matrices for computing Kalman filter state evolution over time
xhat_k_pred = np.zeros((NumSteps,2,1))
xhat_k_curr = np.zeros((NumSteps,2,1))
P_k_pred = np.zeros((NumSteps,2,2))
P_k_curr = np.zeros((NumSteps,2,2))
Kfk_curr = np.zeros((NumSteps,2,1))
I = np.identity(2)

# setup the initial states of the prediction steps, xhat 0|-1 and P 0|-1
xhat_k_pred[0,:,:] = xhat_init
P_k_pred[0,:,:] = P_init[:,:]

# start running the Extended Kalman Filter, using the NoisyMeasurements
for t in np.arange(1,NumSteps):
    ## update step, given the measurement
    # K f,i-1
    Hkm1 = H(xhat_k_pred[t-1,:,:])
    Kfk_curr[t-1,:,:] = P_k_pred[t-1,:,:] @ Hkm1.T @ np.linalg.inv(Hkm1 @ P_k_pred[t-1,:,:] @ Hkm1.T + Rk)
    # P i-1|i-1
    P_k_curr[t-1,:,:] = (I - Kfk_curr[t-1,:,:] @ Hkm1) @ P_k_pred[t-1,:,:]
    # x i-1|i-1
    xhat_k_curr[t-1,:,:] = xhat_k_pred[t-1,:,:] + Kfk_curr[t-1,:,:] * (y[t-1] - h(xhat_k_pred[t-1,:,:]) )

    ## Predicition Step
    # x i|i-1
    xhat_k_pred[t,:,:] = f(xhat_k_curr[t-1,:,:],Delta)
    # P i|i-1
    P_k_pred[t,:,:] = F(xhat_k_curr[t-1,:,:],Delta) @ P_k_curr[t-1,:,:] @ F(xhat_k_curr[t-1,:,:],Delta).T + Qk

## and set the last Update step
t = NumSteps - 1
# K f,t
Hkm1 = H(xhat_k_pred[t,:,:])
Kfk_curr[t,:,:] = P_k_pred[t,:,:] @ Hkm1.T @ np.linalg.inv(Hkm1 @ P_k_pred[t,:,:] @ Hkm1.T + Rk)
# P t|t
P_k_curr[t,:,:] = (I - Kfk_curr[t,:,:] @ Hkm1) @ P_k_pred[t,:,:]
# x t|t
xhat_k_curr[t,:,:] = xhat_k_pred[t,:,:] + Kfk_curr[t,:,:] * (y[t] - h(xhat_k_pred[t,:,:]) )

fig, ax = plt.subplots(figsize=(7,5), dpi=120)

ax.plot(tscale,x,'b')
ax.grid(True)
ax.plot(tscale_measurement,y,'c+')
ax.plot(tscale_measurement,xhat_k_curr[:,0,:],'r-')
ax.legend([r'$x_1$',r'y, $\Delta t = 20ms$',r'$\hat{x}_{1,k|k}$'])
ax.set_ylabel(r'Pendulum angle: $\alpha = x_1$')
ax.set_xlabel(r'Time (seconds)')
fig.suptitle(r'EKF for $\Delta t = 20ms$')

plt.show()

```



4.3.1 RMS for y

In [9]:

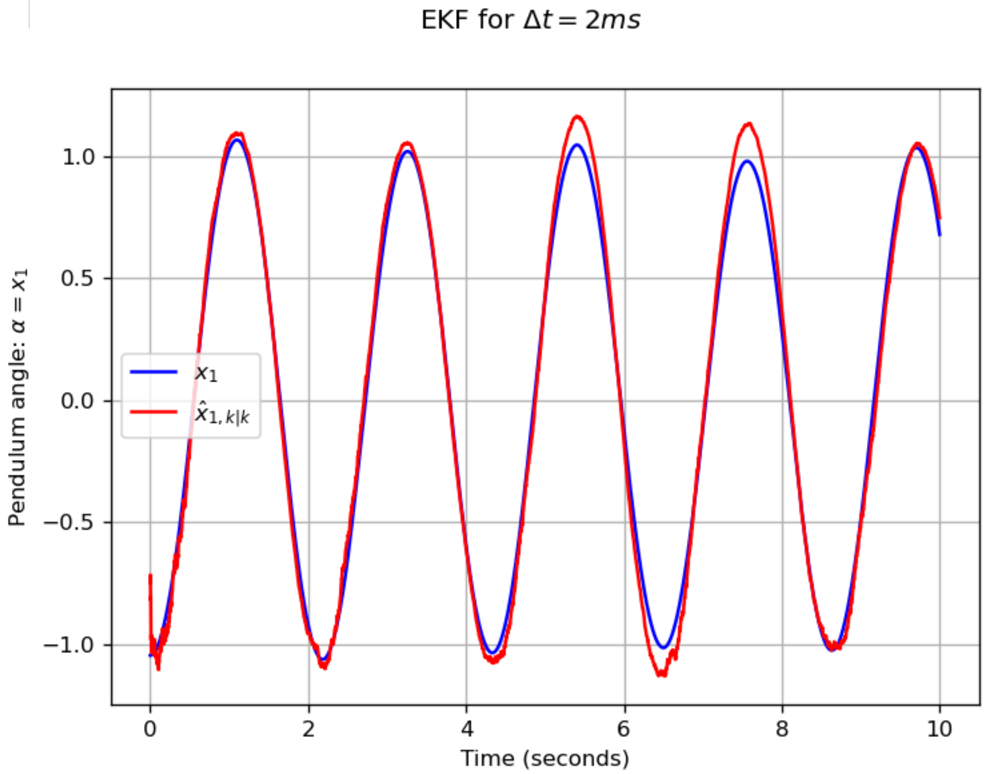
```
x_20ms = x[::20]
error = y - x_20ms
error_kalman_f = xhat_k_curr[:,0,:].reshape(-1) - x_20ms
rms_error = np.sqrt(np.sum(error**2)/len(y))
rms_kalman_f = np.sqrt(np.sum(error_kalman_f**2)/len(y))

print(f'''RMS of the measurement y error: {rms_error}''')
print(f'''RMS of the EKF estimates: {rms_kalman_f}''')
```

```
RMS of the measurement y error: 0.31853835229944727
RMS of the EKF estimates: 0.18003291734915677
```

4.4 Implement the EKF for measurements2.npy

In [10]:

```python
# initialize all of the variables that we're going to need
Delta = 0.002 # 2 ms
sigma_m = 0.3
sigma_p = 0.1
Qk = (sigma_p**2)*np.matrix([[Delta**3/3,Delta**2/2],
                             [Delta**2/2,Delta]])
Rk = np.matrix([sigma_m**2])
g = 9.8 # m/s^2

def f(xk,dt):
    fk_x = np.matrix([[xk[0][0] + xk[1][0]*dt],
                      [xk[1][0] - g*dt*np.sin(xk[0][0])]])
    return fk_x

def F(xhat,dt):
    F_x = np.matrix([[1                       , dt],
                     [-g*dt*np.sin(xhat[0][0]), 1]])
    return F_x

def h(xk,dt=None):
    return np.sin(xk[0][0])

def H(xhat,dt=None):
    return np.matrix([np.cos(xhat[0][0]), 0])

# the initial guesses of x and P
xhat_init = np.array([[y2[0]],[0]]) # set alpha = y2[0] and d(alpha)/dt = 0
P_init = np.identity(2)
NumSteps = len(y2)

# initializing the matrices for computing Kalman filter state evolution over time
xhat_k_pred = np.zeros((NumSteps,2,1))
xhat_k_curr = np.zeros((NumSteps,2,1))
P_k_pred = np.zeros((NumSteps,2,2))
P_k_curr = np.zeros((NumSteps,2,2))
Kfk_curr = np.zeros((NumSteps,2,1))
I = np.identity(2)

# setup the initial states of the prediction steps, xhat 0|-1 and P 0|-1
xhat_k_pred[0,:,:] = xhat_init
P_k_pred[0,:,:] = P_init[:,:]

# start running the Extended Kalman Filter, using the NoisyMeasurements
for t in np.arange(1,NumSteps):
    ## update step, given the measurement
    # K f,i-1
    Hkm1 = H(xhat_k_pred[t-1,:,:])
    Kfk_curr[t-1,:,:] = P_k_pred[t-1,:,:] @ Hkm1.T @ np.linalg.inv(Hkm1 @ P_k_pred[t-1,:,:] @ Hkm1.T + Rk)
    # P i-1|i-1
    P_k_curr[t-1,:,:] = (I - Kfk_curr[t-1,:,:] @ Hkm1) @ P_k_pred[t-1,:,:]
    # x i-1|i-1
    xhat_k_curr[t-1,:,:] = xhat_k_pred[t-1,:,:] + Kfk_curr[t-1,:,:] * (y2[t-1] - h(xhat_k_pred[t-1,:,:]) )

    ## Predicition Step
    # x i|i-1
    xhat_k_pred[t,:,:] = f(xhat_k_curr[t-1,:,:],Delta)
    # P i|i-1
    P_k_pred[t,:,:] = F(xhat_k_curr[t-1,:,:],Delta) @ P_k_curr[t-1,:,:] @ F(xhat_k_curr[t-1,:,:],Delta).T + Qk

## and set the last Update step
t = NumSteps - 1
# K f,t
Hkm1 = H(xhat_k_pred[t,:,:])
Kfk_curr[t,:,:] = P_k_pred[t,:,:] @ Hkm1.T @ np.linalg.inv(Hkm1 @ P_k_pred[t,:,:] @ Hkm1.T + Rk)
# P t|t
P_k_curr[t,:,:] = (I - Kfk_curr[t,:,:] @ Hkm1) @ P_k_pred[t,:,:]
# x t|t
xhat_k_curr[t,:,:] = xhat_k_pred[t,:,:] + Kfk_curr[t,:,:] * (y2[t] - h(xhat_k_pred[t,:,:]) )

fig, ax = plt.subplots(figsize=(7,5), dpi=120)

ax.plot(tscale,x,'b')
ax.grid(True)
# ax.plot(tscale_measurement2,y2,'c+')
ax.plot(tscale_measurement2,xhat_k_curr[:,0,:],'r-')
# ax.legend([r'$x_1$','y2',r'$\hat{x}_{1,k|k}$'])
ax.legend([r'$x_1$',r'$\hat{x}_{1,k|k}$'])
ax.set_ylabel(r'Pendulum angle: $\alpha = x_1$')
ax.set_xlabel(r'Time (seconds)')
fig.suptitle(r'EKF for $\Delta t = 2ms$')

plt.show()
```



4.4.1 RMS for y2

In [11]:

```python
x_2ms = x[::2]
error = y2 - x_2ms
error_kalman_f = xhat_k_curr[:,0,:].reshape(-1) - x_2ms
rms_error = np.sqrt(np.sum(error**2)/len(y2))
rms_kalman_f = np.sqrt(np.sum(error_kalman_f**2)/len(y2))

print(f'''RMS of the measurement y2 error: {rms_error}''')
print(f'''RMS of the EKF estimates: {rms_kalman_f}''')
```

```
RMS of the measurement y2 error: 0.31134887639065506
RMS of the EKF estimates: 0.0677126618156732
```

5 Particle Filter

5.1 The Particle Filter algorithm (with resampling step)

In [12]:

```
Delta = 0.020 # 20 ms
sigma_m = 0.3
sigma_p = 0.1
Qk = (sigma_p**2)*np.matrix([[Delta**3/3,Delta**2/2],
                             [Delta**2/2,Delta]])
NumSteps = len(y)

# 1) draw n samples from the prior
# 2) for each k = 1...T
#     a) draw samples x_k(i) from the importance distribution
#     b) compute the new weights
#     c) normalize the new weights

# initialize x^i_k and w^i_k matrices to keep track of state estimation distributions and weights
n = 200 # number of particles
xki = np.zeros((NumSteps,2,n),dtype=float)
wki = np.zeros((NumSteps,n),dtype=float)
pi_ki = np.zeros((NumSteps,2,n))

# 1) draw n samples from the prior
x0_mu, x0_sigma = y[0], np.sqrt(0.5)
x0 = np.random.normal(x0_mu, x0_sigma, n)
w0 = 1/n*np.ones(n)

# insert the samples from the prior into our matrices for keeping track of things
xki[0,:] = x0
wki[0,:] = w0

# initialize noise Gaussian parameters
q_mu, q_cov = [0,0], Qk
v_mu, v_sigma = 0, sigma_m

# 2) for each k = 1...T
mean = np.zeros(NumSteps) # keep track of the mean of the particles
var = np.zeros(NumSteps) # keep track of the variance of the particles at each step
neff = np.zeros(NumSteps)

mean[0] = x0_mu
var[0] = x0_sigma**2

##################
# Need to figure out how to accurately sample and push 2nd state through the PF as well
# the following code will not function without more work/massaging being done.
##################

# for k in np.arange(1,NumSteps,1):
#     # a) draw samples x_k(i) from the importance distribution
#     # pi_ki[k,:,:] = np.matrix([[]])
#     xki[k,:] = 1/2*xki[k-1,:] + 25*xki[k-1,:]/(1 + xki[k-1,:]**2) + 8*np.cos(1.2*(k-1)) + \
#                 np.random.multivariate_normal(q_mu, q_cov,n)
#     # print(sum(xki[k,:]))
#     # b) compute the new weights
#     wki[k,:] = wki[k-1,:]*1/np.sqrt(2*np.pi)*np.exp(-0.5*(y[k] - 1/20*(xki[k-1,:]**2))**2)
#     # c) normalize the new weights
#     wki[k,:] = wki[k,:]/sum(wki[k,:])
#     mean[k] = np.average(xki[k,:],weights=wki[k,:])
#     var[k] = np.average((xki[k,:] - mean[k])**2,weights=wki[k,:])
#     neff[k] = 1/sum(wki[k,:]**2)
#     # draw new samples if the number of effective weights is < 20
#     if neff[k] < 20:
#         print(f"""Effective particles < 20 for step {k}""")
#         ind = np.argsort(xki[k,:]) # index sort of the particles
#         xki[k,:] = np.take_along_axis(xki[k,:],ind,axis=0)
#         wki[k,:] = np.take_along_axis(wki[k,:],ind,axis=0) # sort the weights according to the particles
#         bins = np.cumsum(wki[k,:]) # bins from which we are going to sample; cumulative sum of the weights
#         uni = np.random.uniform(0,1,n) # uniform distribution used for re-sampling
#         uni2 = np.random.uniform(0,1,n) # secondary random sampling for within bins
#         for i in np.arange(0,n):
#             for j in np.arange(n-1,-1,-1):
#                 if uni[i] >= bins[j]:
#                     xki[k,i] = xki[k,j] + (xki[k,j+1] - xki[k,j])*uni2[i]
#         # and reset the weights:
#         wki[k,:] = w0

# # track mean for later analysis
# mean_pf_resamp = mean
```

5.2 Plot mean and variance superposed to trajectory

In [13]:

```
# # Plot the trajectory
# fig, ax = plt.subplots(figsize=(8,5), dpi=120)

# ax.plot(np.insert(TimeScale,0,0),x,'b--')
# ax.grid(True)
# ax.plot(np.insert(TimeScale,0,0),x,'bs',markersize=6)
# #plt.legend(['line','markers'])
# ax.set_ylabel(r'State $x_k$')
# ax.set_xlabel(r'Time (sample $k$)')
# for k in np.arange(1,NumSteps,1):
#     for i in np.arange(n):
#         if wki[k,i] > 1e-3:
#             ax.plot(k,xki[k,i],'ro',markersize=10*wki[k,i],alpha=0.3)
# ax.plot(mean,'r+')
# ax.fill_between(np.arange(NumSteps), mean-2*np.sqrt(var), mean+2*np.sqrt(var), alpha=0.25, color='r')
# fig.suptitle('Particle Filter with re-sampling')

# plt.show()
```