Keith On . . .
Estimation and Imaging

K.E. Schubert

Founder
Renaissance Research Labs

Associate Professor
Department of Electrical and Computer Engineering
School of Engineering and Computer Science
Baylor University

Contents

List of Figures

Part I

Estimation

Chapter 1

Differential and Difference Equations

1.1 Classical Mechanics

Newton's Second Law gives us:

$$\begin{aligned} F &= \frac{dp}{dt} && (1.1)\\ &= \frac{dm\dot{x}}{dt} && (1.2)\\ &= m\ddot{x} && (1.3) \end{aligned}$$

Friction and drag are negative forces, usually proportional to velocity ($\dot{x}$). Hooke's spring law tells us that the force exerted by a spring is $-kx$. Thus

$$\begin{aligned} m\ddot{x} &= -c\dot{x} - kx + f(t) && (1.4)\\ m\ddot{x} + c\dot{x} + kx &= f(t) && (1.5) \end{aligned}$$

1.2 Electrical Circuits

$$\begin{aligned} v_l + v_r + v_c &= v(t) && (1.6)\\ L\ddot{q} + R\dot{q} + \frac{1}{C}q &= v(t) && (1.7) \end{aligned}$$

1.3 Canonical Forms

1.4 Difference Equations

$$\begin{aligned}
\dot{x} &= Ax && (1.8)\\
x(t) &= e^{At}x_0 && (1.9)\\
&= e^{A(t-t_{k-1}+t_{k-1})}x_0 && (1.10)\\
&= e^{A(t-t_{k-1})+At_{k-1}}x_0 && (1.11)\\
&= e^{A(\Delta t)}e^{At_{k-1}}x_0 && (1.12)\\
&= e^{A(\Delta t)}x_{k-1} && (1.13)
\end{aligned}$$

Want

$$x_k = A_{k-1}x_{k-1} \quad (1.14)$$

Chapter 2

Least Squares

Suppose we want to solve the following system

$$Ax \approx b \tag{2.1}$$

where $A \in \mathbb{R}^{m \times n}$ describes the system, $x \in \mathbb{R}^n$ are the unknowns, and $b \in \mathbb{R}^m$ are the measurements.

There are a lot of ways of solving this, and the type of answer you want is important. Let us make the most common assumption, that being we want the "error" to be as small as possible[1].

$$\hat{x} = \text{argmin}_x \|Ax - b\| \tag{2.2}$$

This means we want the argument (the value of the variable) that minimizes the expression, and we will call that our estimator. To minimize $\|Ax - b\|$ we first note that it will have the same minimum as $\|Ax - b\|^2$. Why do we care? Well it is easier to take the derivative of the squared term. Taking the gradient (vector derivative)

$$\begin{aligned} \nabla_x \|Ax - b\|^2 &= \nabla_x \left((Ax - b)^T (Ax - b)\right) && (2.3)\\ &= \nabla_x \left(x^T A^T Ax - 2b^T Ax + b^T b)\right) && (2.4)\\ &= 2A^T Ax - 2A^T b && (2.5)\\ &= 2A^T (Ax - b) && (2.6) \end{aligned}$$

At the minimum the derivative must be zero thus

$$\begin{aligned} 0 &= 2A^T (Ax - b) && (2.7)\\ 0 &= A^T Ax - A^T b && (2.8)\\ A^T Ax &= A^T b && (2.9)\\ x &= (A^T A)^{-1} A^T b && (2.10) \end{aligned}$$

[1]Note this means we are assuming all the error is in the measurements, while in fact a reasonable part could be in the system.

Note that Eq 2.9 is called the normal equation, and is the worst way to solve this numerically.

- If the problem is dense[2], non-symmetric[3], and well conditioned[4], then Gaussian Elimination (probably with partial or full pivoting) will work well and is the fastest.
- If the problem is dense, non-symmetric, and bad conditioning[5] (though not very bad), then use QR.
- If the problem is dense, symmetric[6], and not very bad conditioning then use Cholesky.
- If the problem is dense and has very bad conditioning, then use the SVD.

2.1 Recursive Least Squares

$$A_{k+1}x_{k+1} = b_{k+1} \tag{2.11}$$

$$A_{k+1} = \begin{bmatrix} A_k \\ a_{k+1}^T \end{bmatrix}, \quad b_{k+1} = \begin{bmatrix} b_k \\ \beta_{k+1} \end{bmatrix} \tag{2.12}$$

$$\begin{bmatrix} A_k \\ a_{k+1}^T \end{bmatrix} x_{k+1} = \begin{bmatrix} b_k \\ \beta_{k+1} \end{bmatrix} \tag{2.13}$$

Thus

$$A_{k+1}^T A_{k+1} x_{k+1} = A_{k+1}^T b_{k+1} \tag{2.14}$$

$$\begin{bmatrix} A_k \\ a_{k+1}^T \end{bmatrix}^T \begin{bmatrix} A_k \\ a_{k+1}^T \end{bmatrix} x_{k+1} = \begin{bmatrix} A_k \\ a_{k+1}^T \end{bmatrix}^T \begin{bmatrix} b_k \\ \beta_{k+1} \end{bmatrix} \tag{2.15}$$

$$(A_k^T A_k + a_{k+1} a_{k+1}^T) x_{k+1} = A_k^T b_k + a_{k+1}\beta_{k+1} \tag{2.16}$$

and

$$x_{k+1} = (A_{k+1}^T A_{k+1})^{-1} A_{k+1}^T b_{k+1} \tag{2.17}$$

$$= (A_k^T A_k + a_{k+1} a_{k+1}^T)^{-1} (A_k^T b_k + a_{k+1}\beta_{k+1}) \tag{2.18}$$

Now if we define

$$P_{k+1} = (A_{k+1}^T A_{k+1})^{-1} \tag{2.19}$$

$$= (A_k^T A_k + a_{k+1} a_{k+1}^T)^{-1} \tag{2.20}$$

[2] A is mostly non-zero.
[3] $A \neq A^T$
[4] That is, the ratio of the largest to smallest singular values of A is near to 1.
[5] This is a fuzzy term deliberately. Roughly if the condition number is between 10^3 and 10^8, I would call it bad conditioning.
[6] $A = A^T$

Then

$$\begin{aligned} P_{k+1}^{-1} &= A_{k+1}^T A_{k+1} && (2.21)\\ &= A_k^T A_k + a_{k+1} a_{k+1}^T && (2.22)\\ &= P_k^{-1} + a_{k+1} a_{k+1}^T && (2.23)\\ x_{k+1} &= P_{k+1} A_{k+1}^T b_{k+1} && (2.24)\\ P_{k+1}^{-1} x_{k+1} &= A_{k+1}^T b_{k+1} && (2.25) \end{aligned}$$

Next take Eq 2.24

$$\begin{aligned} x_{k+1} &= P_{k+1} A_{k+1}^T b_{k+1} && (2.26)\\ &= P_{k+1}(A_k^T b_k + a_{k+1}\beta_{k+1}) && (2.27)\\ &= P_{k+1}(P_k^{-1} x_k + a_{k+1}\beta_{k+1}) && (2.28)\\ &= P_{k+1}((P_{k+1}^{-1} - a_{k+1}a_{k+1}^T)x_k + a_{k+1}\beta_{k+1}) && (2.29)\\ &= P_{k+1}(P_{k+1}^{-1} x_k - a_{k+1}a_{k+1}^T x_k + a_{k+1}\beta_{k+1}) && (2.30)\\ &= x_k + P_{k+1}(a_{k+1}\beta_{k+1} - a_{k+1}a_{k+1}^T x_k) && (2.31)\\ &= x_k + P_{k+1}a_{k+1}(\beta_{k+1} - a_{k+1}^T x_k) && (2.32)\\ &= x_k + K_{k+1}(\beta_{k+1} - a_{k+1}^T x_k) && (2.33) \end{aligned}$$

Our equations for the recursive least squares (information form) become

$$\begin{aligned} x_{k+1} &= x_k + K_{k+1}(\beta_{k+1} - a_{k+1}^T x_k) && (2.34)\\ K_{k+1} &= P_{k+1} a_{k+1} && (2.35)\\ P_{k+1}^{-1} &= P_k^{-1} + a_{k+1} a_{k+1}^T && (2.36) \end{aligned}$$

2.1.1 Example

Create a function in SciLab to implement the information form of the RLS estimator for one step (i.e. you should pass it $P(k)$,$x(k)$,$a(k+1)$, and $b(k+1)$ and it should return $P(k+1)$ and $x(k+1)$). Now generate a random A and x matrix, and calculate a noiseless b. Assume an initial estimate of $P = I$ and $x = 0$ and then do one iteration of RLS for each row of A. Store all the results intermediate estimates, $\hat{x}$, and plot them versus the "true" value of x.

Solution

The code for the RLS function is straightforward to implement and is shown in Code 2.1. The test code has a few things worth mentioning, see Code 2.2. First, the number of rows in A are the number of iterations we can do in our rls algorithm, as we need 1 row per iteration. Second, the "exec" command is needed to load a non-system library. Third, the initial guess of *xest* is stored in the first column, thus since the entire matrix was initialized to zero, the initial condition for *xest* is zero.

I did four runs of the test code and the resulting graphs are in Fig 2.1 and Fig 2.2. Notice that the solution converges to the real value.

Listing 2.1: Code for RLS information function.

```
function [P,x]=rlsi(P,x,a,b)
// inputs:
//    P is the covariance at time k
//    x is the estimate at time k
//    a is the new system row
//    b is the new data row
// outputs:
//    P is the covariance at time k+1
//    x is the estimate at time k+1
K=P\a';
P=P+a'*a;
x=x+K*(b-a*x)
endfunction
```

Listing 2.2: Code to test RLS information function for random matrices without noise.

```
m=1000;
n=2;
A= rand(m,n);
x=rand(n,1);
b=A*x;
xest=zeros(n,m+1);
P=eye(n,n);
exec rls.sci;

for k=1:m
  [P,xest(:,k+1)]=rls(P,xest(:,k),A(k,:),b(k,:));
end

subplot(1,2,1)
plot([1 m+1],[x(1) x(1)],"b-",1:m+1,xest(1,:),"r-")
xtitle("","Iteration","x[1]")
subplot(1,2,1\2)
plot([1 m+1],[x(2) x(2)],"b-",1:m+1,xest(2,:),"r-")
xtitle("","Iteration","x[2]")
```

2.2 Covariance Form

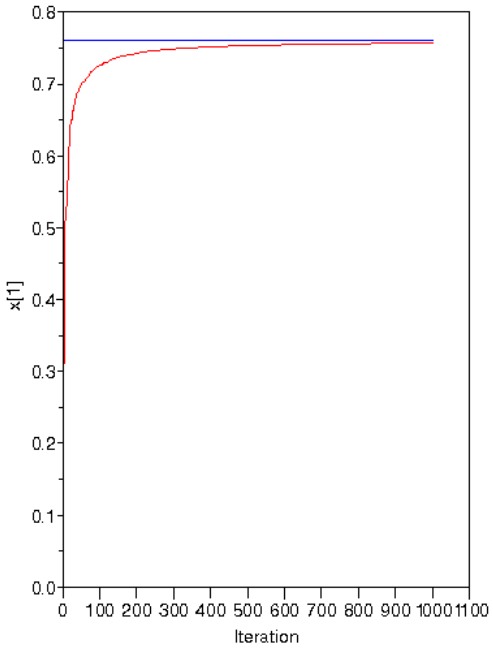
Lemma 1 (Matrix Inversion) *If*

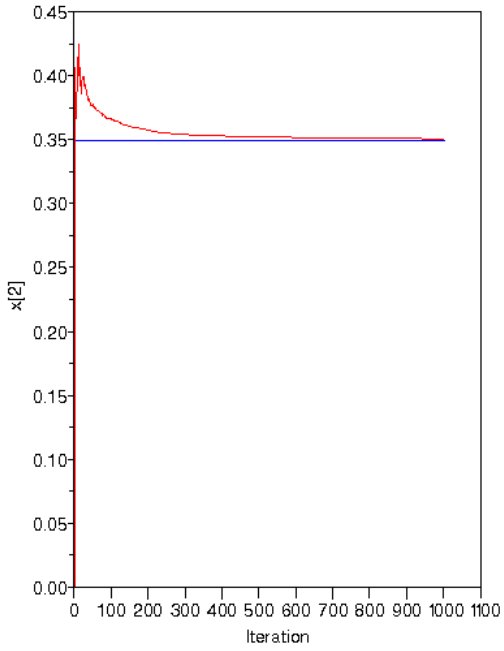
$$A = B + C^T DC \tag{2.37}$$

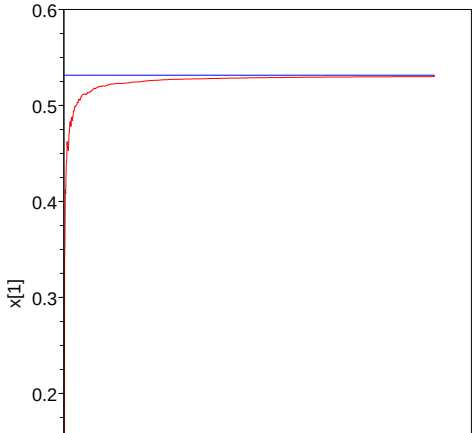
Then

$$A^{-1} = B^{-1} - B^{-1}C^T \left(CB^{-1}C^T + D^{-1}\right)^{-1} CB^{-1} \tag{2.38}$$

Figure 2.1: Comparisons of "True" parameter values versus estimates for several randomly generated problems







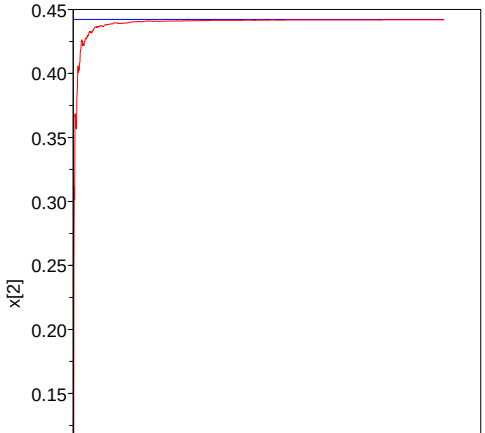
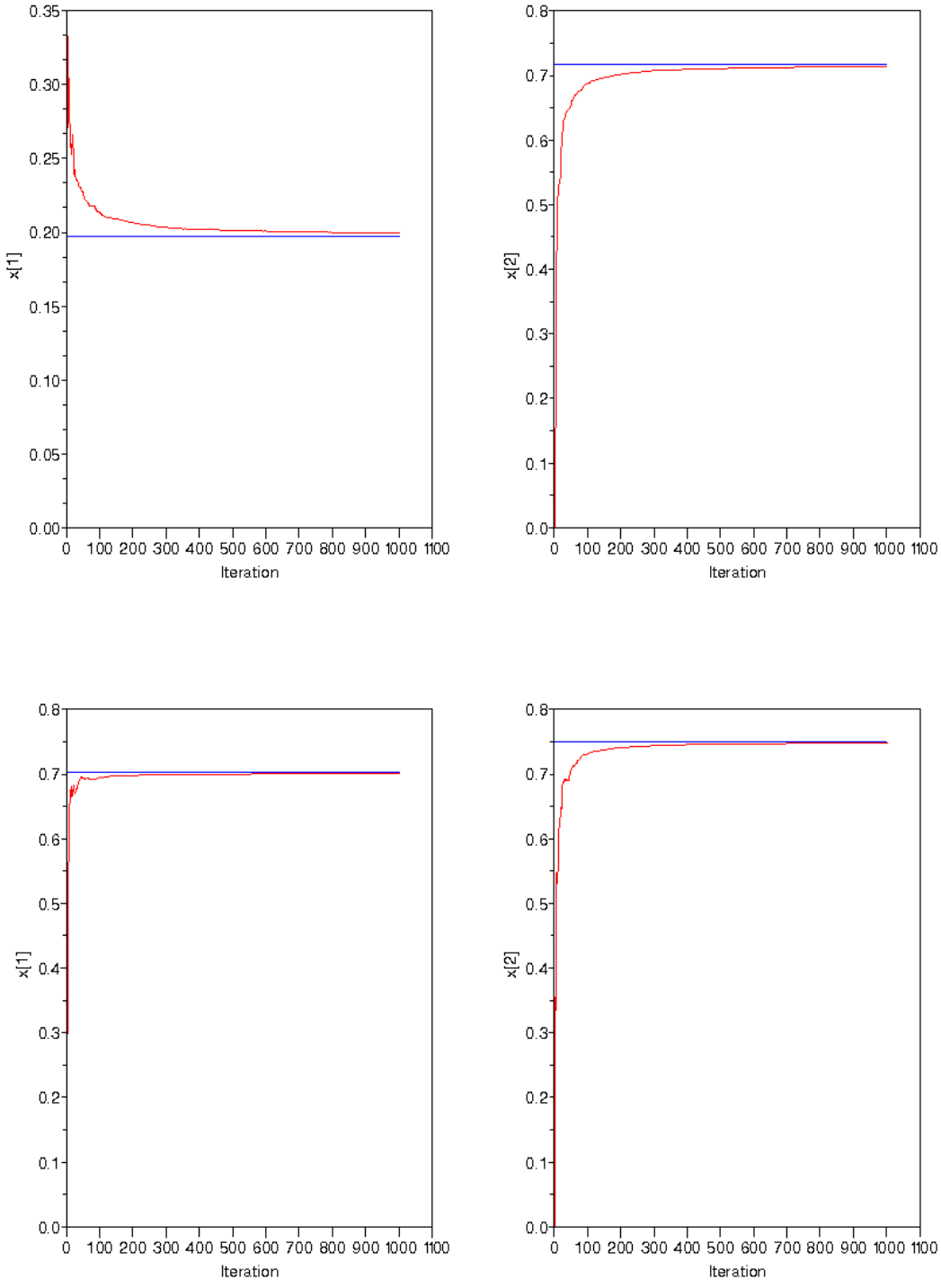


Figure 2.2: More comparisons of "True" parameter values versus estimates for several randomly generated problems



Proof:

$$\begin{aligned}
A &= B + C^T DC \\
I &= A^{-1}B + A^{-1}C^T DC \\
B^{-1} &= A^{-1} + A^{-1}C^T DCB^{-1} \\
B^{-1}C^T &= A^{-1}C^T + A^{-1}C^T DCB^{-1}C^T \\
B^{-1}C^T &= A^{-1}C^T D(D^{-1} + CB^{-1}C^T) \\
B^{-1}C^T(D^{-1} + CB^{-1}C^T)^{-1} &= A^{-1}C^T D \\
B^{-1}C^T(D^{-1} + CB^{-1}C^T)^{-1}CB^{-1} &= A^{-1}C^T DCB^{-1} \\
B^{-1} - B^{-1}C^T(D^{-1} + CB^{-1}C^T)^{-1}CB^{-1} &= B^{-1} - A^{-1}C^T DCB^{-1} \\
B^{-1} - B^{-1}C^T(D^{-1} + CB^{-1}C^T)^{-1}CB^{-1} &= A^{-1} + A^{-1}C^T DCB^{-1} - A^{-1}C^T DCB^{-1} \\
B^{-1} - B^{-1}C^T(D^{-1} + CB^{-1}C^T)^{-1}CB^{-1} &= A^{-1}
\end{aligned}$$

♢ SDG ♢

In our case $A = P_{k+1}^{-1}$, $B = P_k^{-1}$, $C = a_{k+1}^T$, and $D = 1$, thus

$$P_{k+1} = P_k - P_k a_{k+1}\left(a_{k+1}^T P_k a_{k+1} + 1\right)^{-1} a_{k+1}^T P_k \tag{2.39}$$

Now note that

$$\begin{aligned}
K_{k+1} &= P_{k+1}a_{k+1} && (2.40)\\
&= P_k a_{k+1} - P_k a_{k+1}\left(a_{k+1}^T P_k a_{k+1} + 1\right)^{-1} a_{k+1}^T P_k a_{k+1} && (2.41)\\
&= P_k a_{k+1} - P_k a_{k+1}\left(a_{k+1}^T P_k a_{k+1} + 1\right)^{-1}\left(a_{k+1}^T P_k a_{k+1} + 1 - 1\right) && (2.42)\\
&= P_k a_{k+1} - P_k a_{k+1} + P_k a_{k+1}\left(a_{k+1}^T P_k a_{k+1} + 1\right)^{-1} && (2.43)\\
&= P_k a_{k+1}\left(a_{k+1}^T P_k a_{k+1} + 1\right)^{-1} && (2.44)
\end{aligned}$$

Using this we have

$$\begin{aligned}
P_{k+1} &= P_k - K_{k+1}a_{k+1}^T P_k && (2.45)\\
&= (I - K_{k+1}a_{k+1}^T)P_k && (2.46)
\end{aligned}$$

Our equations for the recursive least squares (covariance form) become

$$\begin{aligned}
x_{k+1} &= x_k + K_{k+1}(\beta_{k+1} - a_{k+1}^T x_k) && (2.47)\\
K_{k+1} &= P_k a_{k+1}\left(a_{k+1}^T P_k a_{k+1} + 1\right)^{-1} && (2.48)\\
P_{k+1} &= (I - K_{k+1}a_{k+1}^T)P_k && (2.49)
\end{aligned}$$

2.2.1 Example

Create a function in SciLab to implement the covariance form of the RLS estimator for one step (i.e. you should pass it $P(k)$,$x(k)$,$a(k+1)$, and $b(k+1)$ and it should return $P(k+1)$ and $x(k+1)$). Now generate a random A and x matrix, and calculate a noiseless b. Assume an initial estimate of $P = I$ and $x = 0$ and then do one iteration of RLS for each row of A. Store all the results intermediate estimates, $\hat{x}$, and plot them versus the "true" value of x.

Solution

The code for the RLS function is straightforward to implement and is shown in Code 2.3. The test code has a few things worth mentioning, see Code 2.4. First, the number of rows in A are the number of iterations we can do in our rls algorithm, as we need 1 row per iteration. Second, the "exec" command is needed to load a non-system library. Third, the initial guess of *xest* is stored in the first column, thus since the entire matrix was initialized to zero, the initial condition for *xest* is zero.

I did four runs of the test code and the resulting graphs are in Fig 2.3 and Fig 2.4. Notice that the solution converges to the real value.

Listing 2.3: Code for RLS function.

```
function [P,x]=rls(P,x,a,b)
// inputs:
//    P is the covariance at time k
//    x is the estimate at time k
//    a is the new system row
//    b is the new data row
// outputs:
//    P is the covariance at time k+1
//    x is the estimate at time k+1
K=P*a'./(1+a*P*a');
P=P-K*a*P;
x=x+K*(b-a*x)
endfunction
```

Listing 2.4: Code to test RLS function for random matrices without noise.

```
m=1000;
n=2;
A= rand(m,n);
x=rand(n,1);
b=A*x;
xest=zeros(n,m+1);
P=eye(n,n);
exec rls.sci;

for k=1:m
  [P,xest(:,k+1)]=rls(P,xest(:,k),A(k,:),b(k,:));
```

```
end
subplot(1,2,1)
plot([1 m+1],[x(1) x(1)],"b-",1:m+1,xest(1,:),"r-")
xtitle("","Iteration","x[1]")
subplot(1,2,1\2)
plot([1 m+1],[x(2) x(2)],"b-",1:m+1,xest(2,:),"r-")
xtitle("","Iteration","x[2]")
```

2.3 Estimation with Noise

$$y = Ax + v \tag{2.50}$$

For x and v independent Gaussian random variables. Since y is a linear combination of Gaussian random variables, it is also gaussian. The mean of the measurements is

$$\begin{aligned} E[y] &= E[Ax + v] && (2.51)\\ &= AE[x] + E[v] && (2.52)\\ &= A\bar{x} && (2.53) \end{aligned}$$

and their covariance is

$$\begin{aligned} E[(y - E[y])(y - E[y])^T] &= E[(A(x - \bar{x}) + v)(A(x - \bar{x}) + v)^T] && (2.54)\\ &= E[A(x - \bar{x})(x - \bar{x})^T A^T + vv^T] && (2.55)\\ &= AP_xA^T + P_v. && (2.56) \end{aligned}$$

Now, note the covariance of x and y is

$$\begin{aligned} E[(x - E[x])(y - E[y])^T] &= E[(x - \bar{x})(A(x - \bar{x}) + v)^T] && (2.57)\\ &= E[(x - \bar{x})(x - \bar{x})^T A^T] && (2.58)\\ &= P_xA^T. && (2.59) \end{aligned}$$

Keep these results for later, as I will use them to interpret the resulting filter. We could divide the observation equation up into time instants as we just did, and this would work, however I will take a different approach so you are exposed to several different solution techniques. From our recursive filter above we would like to find a recursive filter that updates based on new (not predictable from past measurements) information in the system. At some time $k+1$, given measurements up to time k, we want a linear estimator, as it will then be a gaussian random variable like the state we wish to estimate. Since we would like to write it as an updating equation, so we can process new data as it arrives, the update must be:

$$\begin{aligned} \hat{x}_{k+1} &= \hat{x}_k + K_k\nu_{k+1|k} && (2.60)\\ \nu_{k+1|k} &= y_{k+1} - A\hat{x}_k && (2.61) \end{aligned}$$

Figure 2.3: Comparisons of "True" parameter values versus estimates for several randomly generated problems

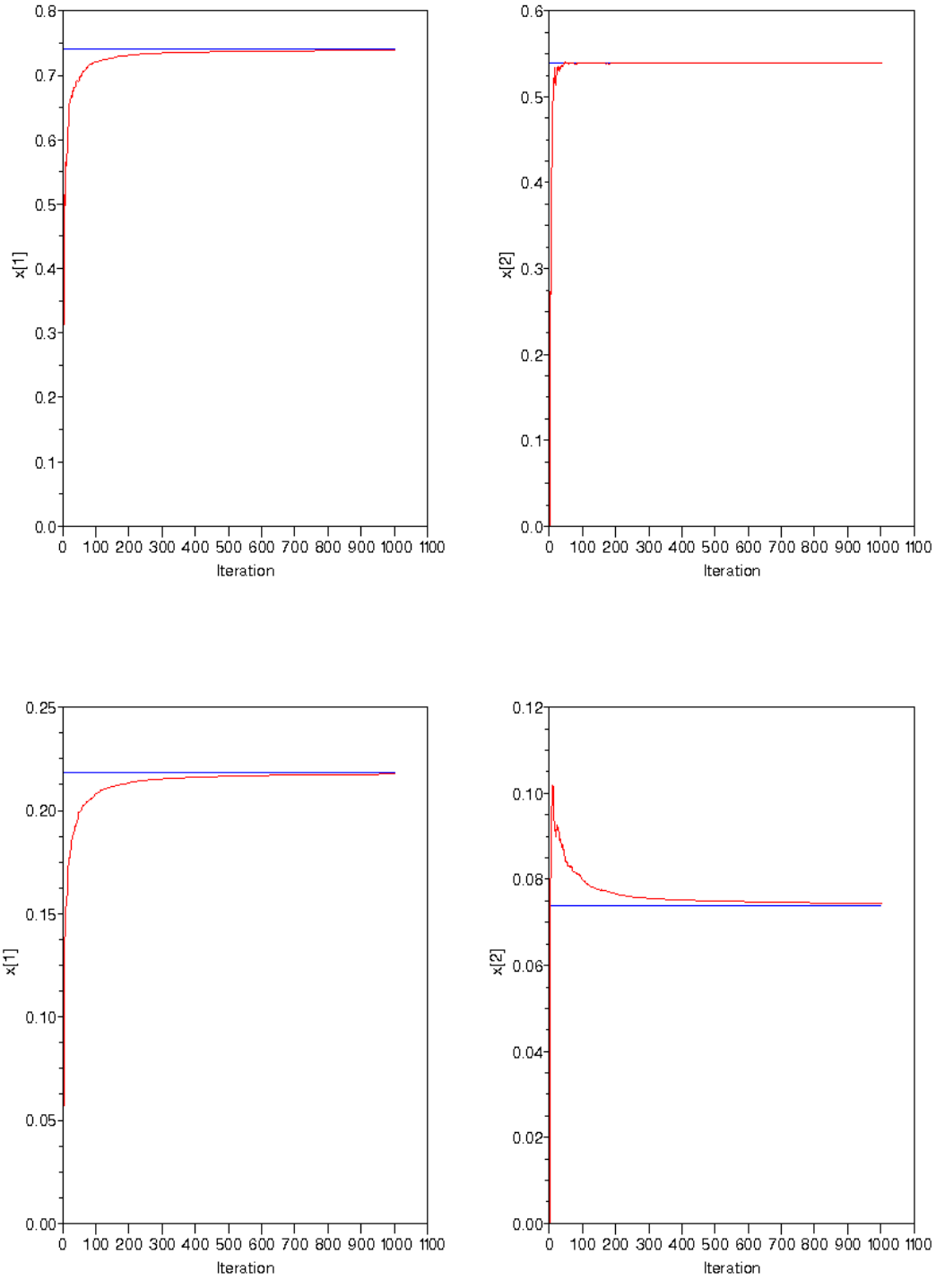
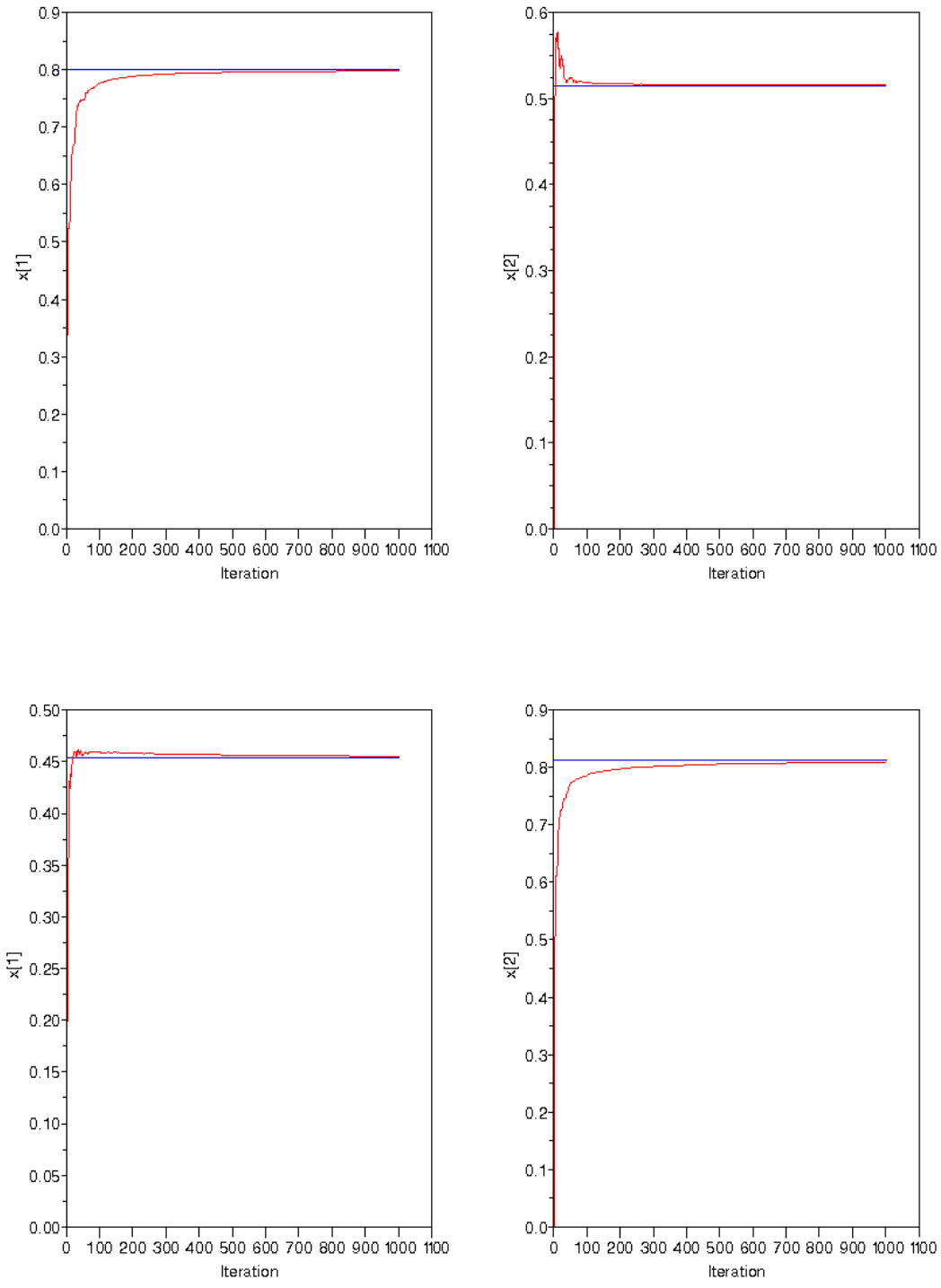


Figure 2.4: More comparisons of "True" parameter values versus estimates for several randomly generated problems



We want the minimum mean square error in $\hat{x}_k \forall k$ so consider the mean of the error and the covariance of the error in x

$$\begin{aligned} P_e k &= E[e_k e_k^T] && (2.62)\\ &= E[(x-\hat{x}_k)(x-\hat{x}_k)^T] && (2.63)\\ &= E[(x-\mu+\mu-\hat{x}_k)(x-\mu+\mu-\hat{x}_k)^T] && (2.64)\\ &= E[(x-\mu)(x-\mu)^T] + E[(x-\mu)(\mu-\hat{x}_k)^T] + E[(\mu-\hat{x}_k)(x-\mu)^T] + E[(\mu-\hat{x}_k)(\mu-\hat{x}_k)^T] && (2.65) \end{aligned}$$

Now

$$\begin{aligned} P_e k+1 &= E[e_{k+1} e_{k+1}^T] && (2.66)\\ &= E[(x-\hat{x}_{k+1})(x-\hat{x}_{k+1})^T] && (2.67)\\ &= E[(x-\hat{x}_k - K_k\nu_{k+1|k})(x-\hat{x}_k - K_k\nu_{k+1|k})^T] && (2.68)\\ &= E[(e_k - K_k\nu_{k+1|k})(e_k - K_k\nu_{k+1|k})^T] && (2.69) \end{aligned}$$

Note that the error, by definition is a gaussian random variable (linear combo of x and $\hat{x}$).

$$\begin{aligned} P_e k+1 &= E[(e_k - K_k\nu_{k+1|k})(e_k - K_k\nu_{k+1|k})^T] && (2.70)\\ &= E[e_k e_k^T] - E[e_k \nu_{k+1|k}^T K_k^T] - E[K_k \nu_{k+1|k} e_k^T] + E[K_k \nu_{k+1|k} \nu_{k+1|k}^T K_k^T] && (2.71)\\ &= P_e k - E[e_k \nu_{k+1|k}^T] K_k^T - K_k E[\nu_{k+1|k} e_k^T] + K_k E[\nu_{k+1|k}\nu_{k+1|k}^T] K_k^T && (2.72) \end{aligned}$$

We need to calculate $E[\nu_{k+1|k} e_k^T]$ and $E[\nu_{k+1|k}\nu_{k+1|k}^T]$

$$\begin{aligned} E[\nu_{k+1|k} e_k^T] &= E[(y_{k+1} - A\hat{x}_k) e_k^T] && (2.73)\\ &= E[(Ax + v_{k+1} - A\hat{x}_k) e_k^T] && (2.74)\\ &= E[(Ae_k + v_{k+1}) e_k^T] && (2.75)\\ &= AP_e k && (2.76) \end{aligned}$$

and

$$\begin{aligned} E[\nu_{k+1|k}\nu_{k+1|k}^T] &= E[(y_{k+1} - A\hat{x}_k)(y_{k+1} - A\hat{x}_k)^T] && (2.77)\\ &= E[(Ax + v_{k+1} - A\hat{x}_k)(Ax + v_{k+1} - A\hat{x}_k)^T] && (2.78)\\ &= E[(Ae_k + v_{k+1})(Ae_k + v_{k+1})^T] && (2.79)\\ &= AP_e k A^T + P_v && (2.80) \end{aligned}$$

Thus,

$$P_e k+1 = P_e k - P_e k A^T K_k^T - K_k A P_e k + K_k(AP_e k A^T + P_v) K_k^T \quad (2.81)$$

Now take the derivative with respect to K_k and set it equal to zero

$$\begin{aligned} 0 &= -2P_e k A^T + 2K_k(AP_e k A^T + P_v) && (2.82)\\ K_k(AP_e k A^T + P_v) &= P_e k A^T && (2.83)\\ K_k &= P_e k A^T (AP_e k A^T + P_v)^{-1} && (2.84) \end{aligned}$$

Thus

$$\begin{aligned} \hat{x}_{k+1} &= \hat{x}_k + K_k(y_{k+1} - A\hat{x}_k) && (2.85)\\ K_k &= P_e k A^T (A P_e k A^T + P_v)^{-1} && (2.86)\\ P_e k + 1 &= P_e k - P_e k A^T K_k^T - K_k A P_e k + K_k (A P_e k A^T + P_v) K_k^T && (2.87) \end{aligned}$$

$$\begin{aligned} E[y - E[\hat{y}]] &= E[A(x - \hat{x}) + v] && (2.88)\\ &= AE[(x - \hat{x})] + E[v] && (2.89)\\ &= A\bar{x} && (2.90) \end{aligned}$$

Thus

$$\begin{aligned} \hat{x} &= E[x|y] && (2.91)\\ &= E[(y - E[y])(y - E[y])^T]^{-1} E[(x - E[x])(y - E[y])^T] && (2.92)\\ &= P_x A^T && (2.93) \end{aligned}$$

2.4 Projections

idempotent

$$P^2 = P \tag{2.94}$$

Example: [Oblique projection] Consider the following:

$$\begin{aligned} P &= \begin{bmatrix} 0 & 0 \\ \alpha & 1 \end{bmatrix} && (2.95)\\ P^2 &= \begin{bmatrix} 0 & 0 \\ \alpha & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ \alpha & 1 \end{bmatrix} && (2.96)\\ &= \begin{bmatrix} 0 & 0 \\ \alpha & 1 \end{bmatrix} && (2.97)\\ &= P && (2.98) \end{aligned}$$

Thus P is idempotent, and thus a projector.

Orthogonal projection

$$P^2 = P = P^T \tag{2.99}$$

Chapter 3

State Observation

3.1 Discrete Time Observability

observability matrix straightforward example of full rank finding initial state

3.2 Discrete Time Detectability

unobservable states are asymptotically stable

3.3 Continuous Time Observability

observability matrix by derivatives example of full rank finding state

3.4 Continuous Time Detectability

3.5 Laplace Domain

$$\dot{x} = Ax + Bu \tag{3.1}$$
$$Y = Cx \tag{3.2}$$

$$sX = AX + BU \tag{3.3}$$
$$Y = CX \tag{3.4}$$

$$
\begin{aligned}
sX - AX &= BU && (3.5)\\
(sI - A)X &= BU && (3.6)\\
X &= (sI - A)^{-1}BU && (3.7)
\end{aligned}
$$

$$
\begin{aligned}
Y &= CX && (3.8)\\
&= C(sI - A)^{-1}BU && (3.9)
\end{aligned}
$$

$$
H(s) = C(sI - A)^{-1}B \qquad (3.10)
$$

Chapter 4

Kalman Filtering

4.1 Random Variables

Say I have a random variable, x, that is normally distributed with mean of $E[x] = \mu$ and variance of $E[(x-\mu)^2] = \sigma^2$. We write this as

$$p(x) \sim N(\mu, \sigma^2). \tag{4.1}$$

Sometimes we use the standard deviation instead of the variance. Note that the standard deviation is the square root of the variance, and is useful because it has the same units as the original variable.

Now let the random variable pass through a linear system described by the equation $y = ax + b$. What is the distribution of y?

The reason we like normal random variables is that linear functions of normally distributed random variables are also normally distributed. The mean of y is just the mean of x passed through the system, $a\mu + b$. To get the variance we find

$$\begin{aligned} E[(y - E[y])^2] &= E[(ax + b - E[ax + b])^2] && (4.2)\\ &= E[(ax + b - a\mu - b)^2] && (4.3)\\ &= E[(a(x-\mu))^2] && (4.4)\\ &= a^2 E[(x-\mu)^2] && (4.5)\\ &= a^2\sigma^2 && (4.6) \end{aligned}$$

Thus $p(y) = N(a\mu + b, a^2\sigma^2)$.

4.2 Discrete Kalman Filter

Assume we have a state space representation

$$\begin{aligned} x_{k+1} &= A_k x_k + B_k u_k + w_k && (4.7)\\ y_k &= C_k x_k + D_k u_k + v_k && (4.8) \end{aligned}$$

Since the control and estimator are independent, we can dump the terms with u Assume we have a state space representation

$$x_{k+1} = A_k x_k + w_k \tag{4.9}$$
$$y_k = C_k x_k + v_k \tag{4.10}$$

The random variables are given by

$$p(w) \sim N(0, R_w) \tag{4.11}$$
$$p(v) \sim N(0, R_v) \tag{4.12}$$

The Kalman filter is a predictor-corrector system.

- Predict $\hat{x}_{k|k-1}$, using Eq. 4.9. This is done prior to receiving the new observation (this is why it is the estimate of x at time k given data up to time $k-1$, i.e. $\hat{x}_{k|k-1}$), so it is the *a priori estimate*.

- Correct $x_{k|k}$, using Eq. 4.10. This is done after (post) receiving the new observation (this is why it is the estimate of x at time k given data up to time k, i.e. $\hat{x}_{k|k}$), so it is the *a posteriori estimate*.

We want an estimator with as small an error as possible, so we define the error as the difference between our estimate and the actual value. Since we have both an a priori and an a posteriori estimate we need an error for both.

$$e_{k|k-1} = \hat{x}_{k|k-1} - x_k \tag{4.13}$$
$$e_{k|k} = \hat{x}_{k|k} - x_k \tag{4.14}$$

which is also a random variable, with zero mean[1] and variance of

$$P_{k|k-1} = E[e_{k|k-1} e_{k|k-1}^T] \tag{4.15}$$
$$P_{k|k} = E[e_{k|k} e_{k|k}^T] \tag{4.16}$$

Thus we propagate x (the mean of our random estimate) and P (the variance of the estimate).

4.2.1 Prediction Step

Considering our state equation

$$x_{k+1} = A_k x_k + w_k. \tag{4.17}$$

[1] it is an unbiased variable, since $\hat{x}$ is an unbiased estimator. The Kalman Filter is also the Best Linear Unbiased Estimator, or BLUE.

The only value we don't know is w_k, which is a random variable. The best estimate of what is happening is the expected value of the state.

$$\begin{aligned}
\hat{x}_{k+1|k} &= E[x_{k+1}|k] && (4.18)\\
&= E[A_k x_k + w_k|k] && (4.19)\\
&= A_k E[x_k|k] + E[w_k] && (4.20)\\
&= A_k \hat{x}_{k|k} + 0 && (4.21)\\
&= A_k \hat{x}_{k|k}. && (4.22)
\end{aligned}$$

The error in this is

$$\begin{aligned}
e_{k+1|k} &= x_{k+1} - \hat{x}_{k+1|k} && (4.23)\\
&= A_k x_k + w_k - A_k \hat{x}_{k|k} && (4.24)\\
&= A_k e_{k|k} + w_k. && (4.25)
\end{aligned}$$

As an interesting side note, look at the expectation of the error

$$\begin{aligned}
E[e_{k+1|k}] &= E[A_k e_{k|k} + w_k] && (4.26)\\
&= E[A_k e_{k|k}] + E[w_k] && (4.27)\\
&= A_k E[e_{k|k}] + 0 && (4.28)\\
&= A_k E[e_{k|k}]. && (4.29)
\end{aligned}$$

Observe that if all the eigenvalues of A_k have magnitude less than 1 then as $k \to \infty$ the expectation of the error goes to zero. Thus it is easy to see the filter is asymptotically unbiased by design.

Now consider the variance of the error

$$\begin{aligned}
P_{k+1|k} &= E[e_{k+1|k} e_{k+1|k}^T] && (4.30)\\
&= E[(A_k e_{k|k} + w_k)(A_k e_{k|k} + w_k)^T] && (4.31)\\
&= E[(A_k e_{k|k} + w_k)(e_{k|k}^T A_k^T + w_k^T)] && (4.32)\\
&= E[A_k e_{k|k} e_{k|k}^T A_k^T + w_k e_{k|k}^T A_k^T + A_k e_{k|k} w_k^T + w_k w_k^T] && (4.33)\\
&= E[A_k e_{k|k} e_{k|k}^T A_k^T] + E[w_k e_{k|k}^T A_k^T] + E[A_k e_{k|k} w_k^T] + E[w_k w_k^T] && (4.34)\\
&= A_k E[e_{k|k} e_{k|k}^T] A_k^T + E[w_k e_{k|k}^T] A_k^T + A_k E[e_{k|k} w_k^T] + E[w_k w_k^T] && (4.35)\\
&= A_k P_{k|k} A_k^T + (0) A_k^T + A_k (0) + R_w && (4.36)\\
&= A_k P_{k|k} A_k^T + R_w && (4.37)
\end{aligned}$$

In the second to last line the expectations are equal zero because the random variables are independent.

We thus have our two equations for the prediction step

$$\begin{aligned}
\hat{x}_{k+1|k} &= A_k \hat{x}_{k|k} && (4.38)\\
P_{k+1|k} &= A_k P_{k|k} A_k^T + R_w && (4.39)
\end{aligned}$$

4.2.2 Correction

Last time we considered the state equation, let us this time consider the observation equation.

$$y_k = C_k x_k + v_k \tag{4.40}$$

Let us again take the expectation to get the estimate of the observations. We will assume we have only data up to time $k-1$, which will be useful later when we want to get the a posteriori estimate due to the a priori estimate.

$$\begin{aligned}
\hat{y}_{k|k-1} &= E[y_k|k-1] && (4.41)\\
&= E[C_k x_k + v_k|k-1] && (4.42)\\
&= E[C_k x_k|k-1] + E[v_k] && (4.43)\\
&= C_k E[x_k|k-1] + 0 && (4.44)\\
&= C_k \hat{x}_{k|k-1} && (4.45)
\end{aligned}$$

Now let's look at the error in the estimate of the observation

$$\begin{aligned}
\nu_{k|k-1} &= y_k - \hat{y}_{k|k-1} && (4.46)\\
&= y_k - C_k \hat{x}_{k|k-1} && (4.47)\\
&= C_k x_k + v_k - C_k \hat{x}_{k|k-1} && (4.48)\\
&= C_k e_{k|k-1} + v_k && (4.49)
\end{aligned}$$

Notice that as $k \to \infty$ the expectation of this error also goes to zero as the expectation of v_k is zero and the expectation of $e_{k|k-1}$ tends to zero. The error in the estimate of the observations is called the innovations, and it tells us what is new in the observations, i.e. what could not be predicted. While we could not know the state error exactly, we can know the observation error since we get y_k albeit corrupted by noise. We can thus get a measure of the error in the state.

Since the observation is a weighted measure of the state error, we can use it to find the corrected state. We really want to invert the effect of C_k, while taking into account the error covariance.

$$\begin{aligned}
\hat{x}_{k|k} &= \hat{x}_{k|k-1} + K_k \nu_{k|k-1} && (4.50)\\
&= \hat{x}_{k|k-1} + K_k (y_k - C_k \hat{x}_{k|k-1}) && (4.51)
\end{aligned}$$

The weighting term K_k is the Kalman gain at time k. To find the Kalman gain, we will consider the error covariance, which means we need the error (a posteriori in this case).

$$\begin{aligned}
e_{k|k} &= x_k - \hat{x}_{k|k} && (4.52)\\
&= x_k - \hat{x}_{k|k-1} - K_k \nu_{k|k-1} && (4.53)\\
&= e_{k|k-1} - K_k \nu_{k|k-1} && (4.54)
\end{aligned}$$

$$
\begin{aligned}
P_{k|k} &= E[e_{k|k}e_{k|k}^T] && (4.55)\\
&= E[(e_{k|k-1} - K_k\nu_{k|k-1})(e_{k|k-1} - K_k\nu_{k|k-1})^T] && (4.56)\\
&= E[e_{k|k-1}e_{k|k-1}^T] - E[e_{k|k-1}\nu_{k|k-1}^T K_k^T] - E[K_k\nu_{k|k-1}e_{k|k-1}^T]\\
&\quad + E[K_k\nu_{k|k-1}\nu_{k|k-1}^T K_k^T] && (4.57)\\
&= P_{k|k-1} - E[e_{k|k-1}(C_k e_{k|k-1} + v_k)^T K_k^T] - E[K_k(C_k e_{k|k-1} + v_k)e_{k|k-1}^T]\\
&\quad + E[K_k(C_k e_{k|k-1} + v_k)(C_k e_{k|k-1} + v_k)^T K_k^T] && (4.58)\\
&= P_{k|k-1} - E[e_{k|k-1}e_{k|k-1}^T C_k^T K_k^T + e_{k|k-1}v_k^T K_k^T]\\
&\quad - E[K_k C_k e_{k|k-1}e_{k|k-1}^T + K_k v_k e_{k|k-1}^T] + E[K_k C_k e_{k|k-1}e_{k|k-1}^T C_k^T K_k^T\\
&\quad + K_k C_k e_{k|k-1}v_k K_k^T + K_k v_k e_{k|k-1}^T C_k^T K_k^T + K_k v_k v_k K_k^T] && (4.59)\\
&= P_{k|k-1} - E[e_{k|k-1}e_{k|k-1}^T C_k^T K_k^T] + E[e_{k|k-1}v_k^T K_k^T] - E[K_k C_k e_{k|k-1}e_{k|k-1}^T]\\
&\quad + E[K_k v_k e_{k|k-1}^T] + E[K_k C_k e_{k|k-1}e_{k|k-1}^T C_k^T K_k^T]\\
&\quad + E[K_k C_k e_{k|k-1}v_k K_k^T] + E[K_k v_k e_{k|k-1}^T C_k^T K_k^T] + E[K_k v_k v_k K_k^T] && (4.60)\\
&= P_{k|k-1} - P_{k|k-1}C_k^T K_k^T + 0 - K_k C_k P_{k|k-1} + 0\\
&\quad + K_k C_k P_{k|k-1}C_k^T K_k^T + 0 + 0 + K_k R_v K_k^T && (4.61)\\
&= P_{k|k-1} - P_{k|k-1}C_k^T K_k^T - K_k C_k P_{k|k-1} + K_k(C_k P_{k|k-1}C_k^T + R_v)K_k^T && (4.62)
\end{aligned}
$$

Now take the derivative with respect to K_k.

$$
\begin{aligned}
\nabla_{K_k}P_{k|k} &= \nabla_{K_k}P_{k|k-1} - \nabla_{K_k}P_{k|k-1}C_k^T K_k^T - \nabla_{K_k}K_k C_k P_{k|k-1}\\
&\quad + \nabla_{K_k}K_k(C_k P_{k|k-1}C_k^T + R_v)K_k^T && (4.63)\\
0 &= 0 - P_{k|k-1}C_k^T - P_{k|k-1}C_k^T\\
&\quad + 2K_k(C_k P_{k|k-1}C_k^T + R_v) && (4.64)\\
K_k(C_k P_{k|k-1}C_k^T + R_v) &= P_{k|k-1}C_k^T && (4.65)\\
K_k &= P_{k|k-1}C_k^T(C_k P_{k|k-1}C_k^T + R_v)^{-1} && (4.66)
\end{aligned}
$$

The value of K_k can now be put back in the original formula.

$$\begin{aligned}
P_{k|k} &= P_{k|k-1} - P_{k|k-1}C_k^T K_k^T - K_k C_k P_{k|k-1} + K_k(C_k P_{k|k-1} C_k^T + R_v)K_k^T && (4.67)\\
&= P_{k|k-1} - P_{k|k-1}C_k^T (C_k P_{k|k-1} C_k^T + R_v)^{-1} C_k P_{k|k-1} \\
&\quad - P_{k|k-1}C_k^T (C_k P_{k|k-1} C_k^T + R_v)^{-1} C_k P_{k|k-1} \\
&\quad + P_{k|k-1}C_k^T (C_k P_{k|k-1} C_k^T + R_v)^{-1} (C_k P_{k|k-1} C_k^T + R_v) \\
&\quad (C_k P_{k|k-1} C_k^T + R_v)^{-1} C_k P_{k|k-1} && (4.68)\\
&= P_{k|k-1} - 2P_{k|k-1}C_k^T (C_k P_{k|k-1} C_k^T + R_v)^{-1} C_k P_{k|k-1} \\
&\quad + P_{k|k-1}C_k^T (C_k P_{k|k-1} C_k^T + R_v)^{-1} C_k P_{k|k-1} && (4.69)\\
&= P_{k|k-1} - P_{k|k-1}C_k^T (C_k P_{k|k-1} C_k^T + R_v)^{-1} C_k P_{k|k-1} && (4.70)\\
&= P_{k|k-1} - K_k C_k P_{k|k-1} && (4.71)\\
&= (I - K_k C_k) P_{k|k-1} && (4.72)
\end{aligned}$$

We thus have three equations for the prediction step

$$\begin{aligned}
K_k &= P_{k|k-1}C_k^T (C_k P_{k|k-1} C_k^T + R_v)^{-1} && (4.73)\\
\hat{x}_{k|k} &= \hat{x}_{k|k-1} + K_k(y_k - C_k \hat{x}_{k|k-1}) && (4.74)\\
P_{k|k} &= (I - K_k C_k) P_{k|k-1} && (4.75)
\end{aligned}$$

or two if you prefer

$$\begin{aligned}
\hat{x}_{k|k} &= \hat{x}_{k|k-1} + P_{k|k-1}C_k^T (C_k P_{k|k-1} C_k^T + R_v)^{-1}(y_k - C_k \hat{x}_{k|k-1}) && (4.76)\\
P_{k|k} &= P_{k|k-1} - P_{k|k-1}C_k^T (C_k P_{k|k-1} C_k^T + R_v)^{-1} C_k P_{k|k-1} && (4.77)
\end{aligned}$$

4.2.3 Putting It All Together

Predict

$$\begin{aligned}
\hat{x}_{k+1|k} &= A_k \hat{x}_{k|k} && (4.78)\\
P_{k+1|k} &= A_k P_{k|k} A_k^T + R_w && (4.79)
\end{aligned}$$

Correct

$$\begin{aligned}
K_k &= P_{k|k-1}C_k^T (C_k P_{k|k-1} C_k^T + R_v)^{-1} && (4.80)\\
\hat{x}_{k|k} &= \hat{x}_{k|k-1} + K_k(y_k - C_k \hat{x}_{k|k-1}) && (4.81)\\
P_{k|k} &= (I - K_k C_k) P_{k|k-1} && (4.82)
\end{aligned}$$

We can also combine the prediction and correction steps

Update1

$$\begin{aligned}
K_k &= PC_k^T (C_k P C_k^T + R_v)^{-1} && (4.83)\\
\hat{x}_{k+1} &= A_k \hat{x}_k + A_k K_k(y_k - C_k \hat{x}_k) && (4.84)\\
P_{k+1} &= (A_k(I - K_k C_k) P_k A_k^T + R_w) && (4.85)
\end{aligned}$$

Update2

$$K_k = (A_k P_k A_k^T + R_w) C_k^T (C_k (A_k P_k A_k^T + R_w) C_k^T + R_v)^{-1} \quad (4.86)$$
$$\hat{x}_{k+1} = A_k \hat{x}_k + K_k (y_k - C_k \hat{x}_k) \quad (4.87)$$
$$P_{k+1} = (I - K_k C_k)(A_k P_k A_k^T + R_w) \quad (4.88)$$

4.3 Steady State Kalman Filter

Notice that the error covariance never is updated from the measurements, which means that we can either pre-calculate it or we can wait till steady state has been achieved. Let us consider the case when steady state is achieved, thus $P_{k+1} = P_k$ and $A_k = A$, $C_k = C$.

$$K = PC^T (CPC^T + R_v)^{-1} \quad (4.89)$$
$$P = A(I - KC)PA^T + R_w \quad (4.90)$$
$$P = A(I - PC^T (CPC^T + R_v)^{-1} C)PA^T + R_w \quad (4.91)$$
$$P = APA^T - APC^T (CPC^T + R_v)^{-1} CPA^T + R_w \quad (4.92)$$
$$P = APA^T - APC^T (CPC^T + R_v)^{-1} (CPC^T + R_v)(CPC^T + R_v)^{-1} CPA^T + R_w \quad (4.93)$$
$$P = APA^T - AK(CPC^T + R_v)KA^T + R_w \quad (4.94)$$
$$(4.95)$$

4.4 Square Root Filter

Kalman's filter as stated has numerical problems, particularly when P becomes non-symmetrical. This was a critical issue for the Apollo program, so Potter designed the first square root filter for the LEM (Lunar Excursion Module) for the special case of scalar updates and no state noise. Thomas Kailath suggested the general form of propagating the square root rather than the whole covariance. Well not really a square root, actually Choleski Factor $R = LL^T$, for each of R_w, R_v, $P_{k|k}$, and $P_{k|k-1}$.

$$P_{k|k} = U_k U_k^T \quad (4.96)$$
$$P_{k|k-1} = S_k S_k^T \quad (4.97)$$
$$R_v^{-1} = L_v^T L_v \quad (4.98)$$
$$R_w = L_w L_w^T \quad (4.99)$$

4.4.1 Prediction

The basic prediction equations are

$$\hat{x}_{k+1|k} = A_k \hat{x}_{k|k} \quad (4.100)$$
$$P_{k+1|k} = A_k P_{k|k} A_k^T + R_w. \quad (4.101)$$

Only the error covariance equation needs to be rewritten. Substitute our Choleski factors.

$$\begin{aligned} P_{k+1|k} &= A_k P_{k|k} A_k^T + R_w && (4.102)\\ S_{k+1}S_{k+1}^T &= A_k U_k U_k^T A_k^T + L_w L_w^T && (4.103)\end{aligned}$$

Now if, $L_w = 0$ (the original assumption of Potter), then

$$\begin{aligned} S_{k+1}S_{k+1}^T &= A_k U_k U_k^T A_k^T && (4.104)\\ S_{k+1}S_{k+1}^T &= (A_k U_k)(A_k U_k)^T && (4.105)\\ S_{k+1} &= A_k U_k && (4.106)\end{aligned}$$

If not then

$$\begin{aligned} S_{k+1} &= cholesky(A_k U_k U_k^T A_k^T + L_w L_w^T) && (4.107)\\ &= A_k U_k + \Delta L_w && (4.108)\end{aligned}$$

where this is just an update to Potters form. There are a number of algorithms to find ΔL_w, from very fast ones for low rank updates, to ones comparable in complexity to the other updates for full rank. See my section on Choleski for details.

4.4.2 Correction

$$\begin{aligned} K_k &= P_{k|k-1}C_k^T(C_k P_{k|k-1} C_k^T + R_v)^{-1} && (4.109)\\ \hat{x}_{k|k} &= \hat{x}_{k|k-1} + K_k(y_k - C_k\hat{x}_{k|k-1}) && (4.110)\\ P_{k|k} &= (I - K_k C_k)P_{k|k-1} && (4.111)\end{aligned}$$

$$\begin{aligned} K_k &= S_k S_k^T C_k^T (C_k S_k S_k^T C_k^T + (L_v^T L_v)^{-1})^{-1} && (4.112)\\ \hat{x}_{k|k} &= \hat{x}_{k|k-1} + K_k(y_k - C_k\hat{x}_{k|k-1}) && (4.113)\\ U_k U_k^T &= (I - K_k C_k)S_k S_k^T && (4.114)\end{aligned}$$

Consider the last equation.

$$\begin{aligned} U_k U_k^T &= (I - S_k S_k^T C_k^T (C_k S_k S_k^T C_k^T + (L_v^T L_v)^{-1})^{-1} C_k) S_k S_k^T && (4.115)\\ &= S_k (I - S_k^T C_k^T (C_k S_k S_k^T C_k^T + (L_v^T L_v)^{-1})^{-1} C_k S_k) S_k^T && (4.116)\end{aligned}$$

Now use the matrix inversion lemma

$$U_k U_k^T = S_k (I + S_k^T C_k^T L_v^T L_v C_k S_k)^{-1} S_k^T \quad (4.117)$$

Consider the equation for K

$$K_k = S_k S_k^T C_k^T (C_k S_k S_k^T C_k^T + (L_v^T L_v)^{-1})^{-1} \quad (4.118)$$

and define

$$G_k = S_k^T C_k^T \tag{4.119}$$

Then

$$K_k = S_k G_k (G_k^T G_k + L_v^{-1} L_v^{-T})^{-1} \tag{4.120}$$

In potter's case, he was only doing a single measurement so what is inside the parenthesis is just a scalar. and in this case G_k is a vector so this just becomes $d = \|G_k\|^2 + l_v^{-2}$. Then $K_k = \frac{1}{d} S_k G_k$.

4.5 Paige's Filter

The square root filter improves things significantly, but it is not numerically stable. There is a stable version of Kalman's filter, Paige's filter[2].

We will start from the Choleski factors of the last section.

$$P_{k|k}^{-1} = U_k^T U_k \tag{4.121}$$
$$P_{k|k-1} = S_k S_k^T \tag{4.122}$$
$$R_v^{-1} = L_v^T L_v \tag{4.123}$$
$$R_w = L_w L_w^T \tag{4.124}$$

4.5.1 Correction

Recall that the error is a zero mean process with variance of $P_{k|k-1}$ Thus by definition

$$e_{k|k-1} = S_k \zeta_a(k) \tag{4.125}$$

where $\zeta_a(k)$ is a zero mean, unit covariance Gaussian distributed stochastic variable. Doing some algebra we obtain an odd but useful formula.

$$e_{k|k-1} = S_k \zeta_a(k) \tag{4.126}$$
$$\hat{x}_{k|k-1} - x_k = S_k \zeta_a(k) \tag{4.127}$$
$$S_k^{-1} \hat{x}_{k|k-1} = S_k^{-1} x_k + \zeta_a(k) \tag{4.128}$$

Similarly we can write the expression for the observations

$$y_k = C_k x_k + v_k \tag{4.129}$$
$$L_v y_k = L_v C_k x_k + L_v v_k \tag{4.130}$$
$$L_v y_k = L_v C_k x_k + \zeta_b(k) \tag{4.131}$$

[2]Actually it was developed by Paige and Saunders, but it is too long for most people to add Saunders' name.

where $\zeta_b(k)$ is a zero mean, unit variance Gaussian distributed stochastic variable. We can combine these two formulas as follows.

$$\begin{bmatrix} S_k^{-1}\hat{x}_{k|k-1} \\ L_v y_k \end{bmatrix} = \begin{bmatrix} S_k^{-1} \\ L_v C_k \end{bmatrix} x_k + \zeta(k) \tag{4.132}$$

where $\zeta(k)$ is an appropriately sized zero mean, unit variance, Gaussian distributed stochastic variable. Note that this is in the form $b = Ax + \nu$, which means that we can solve for the best estimate of x_k given the prediction and the new data point, i.e. $\hat{x}_{k|k}$. Let's solve this using **QR**.

$$Q^T \begin{bmatrix} S_k^{-1}\hat{x}_{k|k-1} \\ L_v y_k \end{bmatrix} = Q^T \begin{bmatrix} S_k^{-1} \\ L_v C_k \end{bmatrix} \hat{x}_{k|k} \tag{4.133}$$

$$= \begin{bmatrix} U_k \\ 0 \end{bmatrix} \hat{x}_{k|k} \tag{4.134}$$

Part II

Imaging

Chapter 5

Intensity

5.1 Basic Intensity Transforms

The most basic intensity transform we can speak of uses only the intensity of a pixel $I(x, y)$ to determine the value of the transformed pixel at the same location $G(x, y) = T(I(x, y))$. Consider for instance the image negative, $N(x, y)$, of an image, $I(x, y)$ which has levels $[0, n-1]$ is

$$N(x, y) \quad = \quad (n-1) - I(x, y) \tag{5.1}$$

Note that $N(x, y)$ also has levels $[0, n-1]$, and that$N(x, y) + I(x, y) = n - 1$, and is thus the negative (or complement or inverse) in the reduced radix sense (see KOHW: Keith on Hardware for a discussion on radix and reduced radix complement).

5.2 Convolution Masks

5.3 Edge Detectors

Let us begin by noting some properties of edges

1. Edges have a sudden change in pixel values, thus the slope is large. Differentiation will thus show the change.
2. Edges are high frequency phenomena - sudden changes require high frequency components, while gradual changes have mostly low frequency components. This means one way to detect edges would be to use a high pass filter on the image.
3. Edges are continuous so we can "walk the edge" using a left-right technique.

The first two ideas are very similar, in that differentiation is a high pass filter[1].

[1]This is not surprising, since constants (dc) go to zero and monomials drop one degree (lower rate of change).

5.3.1 Differentiation

$$derivative = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \tag{5.2}$$

Sobel

$$S_0 = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \tag{5.3}$$

$$S_1 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{5.4}$$

5.3.2 High Pass Filters

$$Laplacian1 = \begin{bmatrix} -1 & 0 & -1 \\ 0 & 4 & 0 \\ -1 & 0 & -1 \end{bmatrix} \tag{5.5}$$

$$Laplacian2 = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \tag{5.6}$$

$$sharpen = \begin{bmatrix} 0 & -1 & 0 \\ -1 & a & -1 \\ 0 & -1 & 0 \end{bmatrix} \tag{5.7}$$

$$a \in \{5, 6, 7\} \tag{5.8}$$

Chapter 6

Histograms

Each pixel in our image has an intensity associated with it. If we grouped the pixels from our image into bins of similar intensities, then counted them and put the counts into an array (or vector) ordered by the bin number, the result would be a histogram. If you took the histogram and divided each bin count by the total number of pixels in the image, you would have the frequency or probability that a pixel of that intensity bin would be drawn at random from the image. The normalized histogram[1] is thus a discrete approximation of a probability density function (pdf). If we were to make an array the same size as the histogram, but whose j^{th} bin was the sum of the normalized histogram's first j bins, the result would be a discrete approximation of the cumulative density function (cdf). We can also consider histograms, and the associated "pdf" and "cdf" for regions of an image. These region based histograms are statistical samples, and we can use the results of probability and statistics to draw conclusions and examine results.

The rest of this chapter will be examining what we can do with histograms. First we should discuss how we could implement a histogram. Afterwords we will examine how histograms can be used to do intensity correction, such as white-balancing, and finding regions of interest.

6.1 Implementation

To calculate a histogram you must have a region you want to count and then you need to distribute the elements into their respective piles. Consider code 6.1.

Listing 6.1: Histogram for Greyscale Image with Arbitrary Shapes.

```
// histogram_grey
//
// hist=histogram_grey(A, levels , level_list , mask)
```

[1] A histogram divided by the total number of pixels in the image is normalized in the 1-norm since, since its 1-norm is then 1.

```
//  A = matrix of greyscale values to do a histogram on
//  levels = (optional) number of levels to divide histogram into
//  level_list = (optional) list of grey values to divide into
//  mask = (optional) region to consider around a point, allows
//         for non-rectangular regions
//
// calculates a histogram on a region of greyscale values
function hist=histogram_grey(A,levels,level_list,mask)

  [rows,cols]=size(A);

  //  since people don't have to give levels, or
  //  level_list, we must find out what they passed
  //  and pick good values for it.
  if ~exists('level_list') then
    disp('hi')
    if ~exists('levels') then
      levels=2;
    end
    if levels<2 then
      levels=2;
    end
    min_A=min(A);
    max_A=max(A);
    level_list=min_A:(max_A-min_A)/(levels-1):max_A;
  else
    if max(size(level_list))~=levels | norm(level_list)<%eps then
      min_A=min(A);
      max_A=max(A);
      if min_A==max_A then
        levels=1;
        level_list=min_A;
      else
        level_list=min_A:(max_A-min_A)/(levels-1):max_A;
      end
    end
  end
  // set up the mask if a valid one wasn't sent
  if ~exists('mask') then
    mask=ones(A);
  end
  if norm(mask,1)<1 or size(A)~=size(mask) then
    mask=ones(A);
  end
  hist=zeros(level_list);
```

```
  // calculate the histogram
  for i=1:rows
    for j=1:cols
      if mask(i,j)>0 then
        loc=find_location(A(i,j),level_list)
        hist(loc)=hist(loc)+1;
      end
    end
  end
endfunction
```

To do this we need to have a function to find which bin in the histogram it is closest to. Consider code 6.2. Note we have included a binary search version, which is faster since the bins are ordered.

Listing 6.2: Closest Bin Locator.

```
// find_location
//
// location=find_location(key,val_list)
//   key = what you are looking for
//   val_list = ordered list of values to look in
//   location = index of the closest item in the list
//
//  Note the item does not have to be in the list, it will
//  pick the closest item to the list.  Also the list must
//  be ordered small to large.
function location=find_location(key,val_list)
  len=max(size(val_list));
  left=1;
  right=len;
  location=0;

  // check that it is in range
  if key<val_list(left) then
    location=left;
    right=left;
  end
  if key>val_list(right) then
    location=right;
    left=right;
  end

  //find the interval
  while left+1<right
```

```
    mid=floor((left+right)/2);
    if val_list(mid)<key then
      left=mid;
    elseif val_list(mid)>key then
      right=mid;
    else
      location=mid;
      left=right;
    end
  end

  //find the closest location
  if location==0 then
    if left==right then
      location=right;
    else
      if(key-val_list(left))<(val_list(right)-key) then
        location=left;
      else
        location=right;
      end
    end
  end
endfunction
```

6.2 White-balance

The cdf is a monotonically increasing function with a range of 0 to 1. Since it encodes the distribution of pixels with a particular intensity, it can thus be used to balance out the darkness and lightness of a picture, so that the average intensity is 0.5. To prove this consider a transform, $T(\cdot)$, from one random variable to another. Let the original random variable be A and the new one be B. We thus have that

$$|pdf_B dB| = |pdf_A dA| \tag{6.1}$$

$$pdf_B = pdf_A \left|\frac{dA}{dB}\right| \tag{6.2}$$

and since $B(a) = T(A) = \int_{-\infty} apdf_A dA$

$$
\begin{aligned}
pdf_B &= pdf_A \left| \left(\frac{dB}{dA} \right)^{-1} \right| && (6.3)\\
&= pdf_A \left| \left(\frac{dT(A)}{dA} \right)^{-1} \right| && (6.4)\\
&= pdf_A \left| \left(\frac{d \int_{-\infty} apdf_A dA}{dA} \right)^{-1} \right| && (6.5)\\
&= pdf_A \left| (pdf_A)^{-1} \right| && (6.6)\\
&= \frac{pdf_A}{pdf_A} && (6.7)\\
&= 1 && (6.8)
\end{aligned}
$$

Thus the pdf of B is uniform on the interval 0 to 1, and is thus white-balanced. The code in Listing 6.3 shows a MatLab implementation of this method. Note the MatLab code uses the **hist** function, which operates on vectors (matrices are treated as an array of vectors and processed as separte columns). To avoid the problem the matrix is reshaped into a row vector.

Listing 6.3: MatLab code to white-balance an image.

```
function B=hist_correct(A,num_of_bins)
    [A_rows,A_cols]=size(A);
    A_histogram=hist(reshape(A,1,A_rows*A_cols),num_of_bins);
    A_pdf=A_histogram/(A_rows*A_cols);
    A_cdf=A_pdf;
    for bin=2:num_of_bins
        A_cdf(bin)=A_cdf(bin-1)+A_pdf(bin);
    end
    A_cdf=[0 A_cdf];
    min_intensity=min(min(A));
    max_intensity=max(max(A));
    step_intensity=(max_intensity-min_intensity)/num_of_bins;
    intensity_range=min_intensity:step_intensity:max_intensity;

    B=A;
    for row=1:A_rows
        for col=1:A_cols
            intensity_bin=1;
            for bin=2:num_of_bins
                if A(row,col)>intensity_range(bin)
                    intensity_bin=bin;
```

```
                end
            end
            B(row,col)=(A_cdf(intensity_bin)+A_cdf(intensity_bin+1))/2;
        end
    end
    B=mat2gray(B,[min_intensity max_intensity]);
end
```

In practice you might want a different pdf or might have a different interval range, and it is possible to convert to the desired pdf, similarly to what is done here.

6.3 Finding Regions of Interest

Regions which catch our eyes tend to have different histogram distributions than overall one for the entire image. We can thus compare the normalized histogram (pdf) for the image and the normalized histogram for a region and if they are more different than a threshold in some norm sense then we mark the area as interesting and if not, then we don't. We will use relative error to do the comparison, and note that you need to pick the threshold based on the norm you pick. Note this does not guarantee it catches everything, but it will catch that which does not follow the distribution of the rest of the image.

6.3.1 MatLab Implementation

In my MatLab implementation I will again use the **hist** function, which requires the image to be reshaped as a vector. I will further use the two norm to compare. Note that the histograms for each of the regions is quite time consuming, so don't make the radius too big, but also note more bins requires a bigger radius to have pixels to fill them. Two or three works well for the radius and around ten is good for bins (levels). Note also I am ignoring a strip around the border that is the width of region radius to simplify the code (avoids handling either padding or wrapping the image). This is not a difficult task, but makes the code look uglier. The SciLab code handles this and you can see the complexity increase to do so. One final oddity for MatLab programmers, the reshape command on the histogram appears to be unneeded since it is the correct shape, but it is needed since it came from a hyper-matrix the size will be 1 row, 1 column, and then the number of levels/bins in depth. The 1 row and 1 column can clearly be ignored by us but not by a programming language the reshape makes the size just a single row vector and thus avoids a type error.

Listing 6.4: Histogram Region Distinguisher.

```
function map=hist_significant(A,threshold,num_levels,radius)
    [rows,cols]=size(A);
    map=zeros(rows,cols);
    pdf_A=hist(reshape(A,1,rows*cols),num_levels)/(rows*cols);
    rows_reg=rows-2*radius;
    cols_reg=cols-2*radius;
```

```
    pdf_region=zeros(rows_reg,cols_reg,num_levels);
    size_region=(1+2*radius)^2;
    for row=1:rows_reg
        for col=1:cols_reg
            region=A(row:row+2*radius,col:col+2*radius);
            hist_region=hist(reshape(region,1,size_region),num_levels);
            pdf_region(row,col,:)=hist_region/size_region;
        end
    end
    norm_pdf_A=norm(pdf_A);
    for row=radius+1:rows-radius
        for col=radius+1:cols-radius
            pdf_reg=reshape(pdf_region(row-radius,col-radius,:),1,num_levels);
            if norm(pdf_reg-pdf_A)/norm_pdf_A>threshold
                map(row,col)=1;
            end
        end
    end
```

6.3.2 SciLab Implementation

Consider code 6.5.

Listing 6.5: Histogram Region Distinguisher.

```
//  hist_mark
//
//  mask=hist_mark(A,region,threshold,wrap)
//    A = (required)a matrix you wish to be checked
//    region = (optional) a matrix describing the shape of the
//             region to check around each point the entries
//             should be
//                    <=0          -> not part of shape
//                    >0           -> part of shape
//                    max magnitude -> central pixel of region
//              note that if the maximum magnitude item is
//              negative it will not be in the shape
//    threshold = (optional) the cutoff value for what is
//                significance as a relative error of the
//                average histogram.  values between 0 and
//                1, works well around .3 to .5
//    wrap = (optional) a flag, 0 means false, that specifies if
//           the top and bottom, as well as the left and right
//           should wrap around like a torus
//    levels = (optional) how many levels(bins) to use in the
```

```
//              histogram
//    mask = matrix of zeros and ones where the 1's note the
//           pixels of interest
//

function mask=hist_mark(A,region,threshold,wrap,levels)
  mask=zeros(A);
  row=1;
  column=2;
  [rows_A,cols_A]=size(A);

  //the maximum magnitude value of region is where the
  //square is considered to be.  If region is undefined
  // then it becomes a 3x3 square
  if ~exists('region') then
    region=[1 1 1
            1 2 1
            1 1 1];
  end
  // set parameters about the region to check around a point
  [max_val, max_loc]=max(region);
  [rows_region,cols_region]=size(region);
  left=max_loc(column)-1;
  right=cols_region-max_loc(column);
  top=max_loc(row)-1;
  bottom=rows_region-max_loc(row);
  size_region=histogram_grey((region>0).*1,2,1,ones(region));

  if ~exists('levels') then
    levels=2;
  end

  min_A=min(A);
  max_A=max(A);
  level_list=min_A:(max_A-min_A)/(levels-1):max_A;
  hist_A=histogram_grey(A,levels,level_list,ones(A));
  hist_A=hist_A./(rows_A*cols_A);

  // If I am supposed to wrap then copy the overlapping regions
  // to do this easier.  Note we only need to wrap enough to have
  // a margin of top, bottom, left, and right around.  Also note
  // that top wraps to the bottom, left to right, and vice-versa.
  if exists('wrap') then
    if wrap>0 then
      ul=A(rows_A-top+1:rows_A,cols_A-left+1:cols_A);
```

```
      um=A(rows_A-top+1:rows_A,:);
      ur=A(rows_A-top+1:rows_A,1:right);
      ml=A(:,cols_A-left+1:cols_A);
      mr=A(:,1:right);
      ll=A(1:bottom,cols_A-left+1:cols_A);
      lm=A(1:bottom,:);
      lr=A(1:bottom,1:right);
      A=[ul um ur
         ml  A mr
         ll lm lr];
    end
  end

  // my guess as to a reasonable value, probably needs tuning
  if ~exists('threshold') then
    threshold=.5;
  end
  threshold=threshold*levels;

  // calculate mask
  for i=top+1:size(A,row)-bottom
    for j=left+1:size(A,column)-right
      region_A=A(i-top:i+bottom,j-left:j+right);
      hist_pt=histogram_grey(region_A,levels,level_list,region);
      hist_pt=hist_pt./size_region;
      if norm((hist_pt-hist_A)./hist_A)>threshold then
        mask(i,j)=1;
      end
    end
  end

endfunction
```

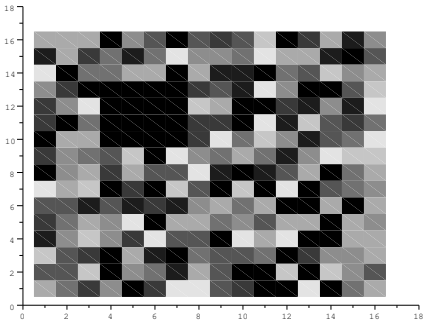
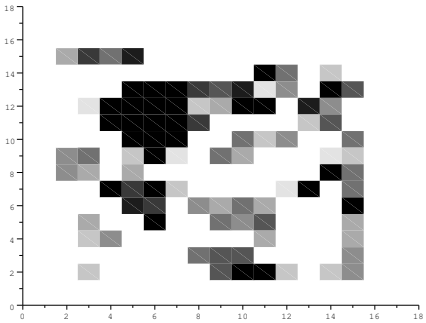
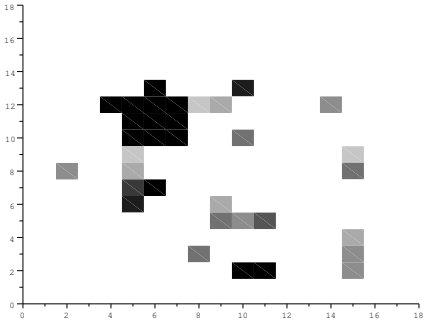
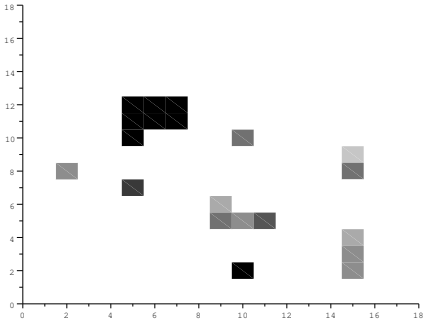
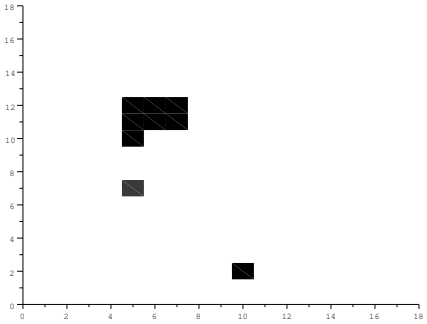
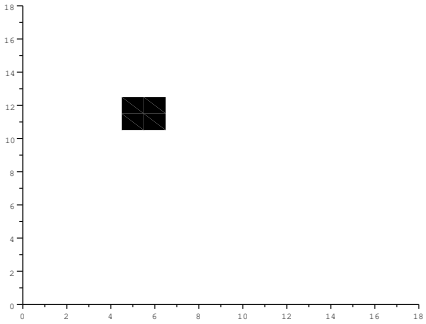
Now that we have code that identifies regions of interest, we need to use it. I used SciLab's "Matplot" command to generate the plots in Figure 6.1. I made the image using "rand" but put in a region of ones so there would be something to find. The command to set up the image were (in this case for threshold=0.3)

```
A=rand(20,20);
A(4:7,4:7)=ones(4,4);
```

The specific commands to generate each picture were

```
f=scf();
Matplot(10-A.*hist_mask(A,threshold=0.3).*10);
f.colormap=graycolormap(10);
```

Table 6.1: Histogram thresholds to yield regions of interest.

Original	Threshold =0.1
	
Threshold = 0.3	Threshold =0.4
	
Threshold = 0.5	Threshold =0.6
	

Chapter 7

Filtering

7.1 Inverse Filter

Probably one of the worst performing filters, but it provides a simple introduction into other filters. The essential idea is that there is some distortion, $h(k,l)$, that has corrupted the image, $x(k,l)$, that has resulted in a distorted image, $y(k,l)$, through convolution,

$$y(k,l) = h(k,l) * x(k,l) \tag{7.1}$$

or in the frequency domain,

$$Y(u,v) = H(u,v)X(u,v). \tag{7.2}$$

The best estimate, $\hat{X}(u,v)$, if there is no noise, the system is known perfectly, and you can calculate with infinite precision is just the inverse of the distortion,

$$\hat{X}(u,v) = H^{-1}(u,v)Y(u,v). \tag{7.3}$$

Technically, we do a pseudoinverse[1], because it is computable and as close to a true inverse as a computer can get. In the ideal situation this means the estimate is

$$\begin{aligned} \hat{X}(u,v) &= H^{-1}(u,v)Y(u,v) && (7.4)\\ &= H^{-1}(u,v)H(u,v)X(u,v) && (7.5)\\ &= X(u,v). && (7.6) \end{aligned}$$

The estimate approaches the actual image. What if there was noise? With noise the system is then $Y(u,v) = H(u,v)X(u,v) + N(u,v)$, and the estimate

$$\begin{aligned} \hat{X}(u,v) &= H^{-1}(u,v)Y(u,v) && (7.7)\\ &= H^{-1}(u,v)\left(H(u,v)X(u,v) + N(u,v)\right) && (7.8)\\ &= X(u,v) + H^{-1}(u,v)N(u,v). && (7.9) \end{aligned}$$

[1]There can be zeros we need to "invert", which would force the inverse to infinity. The pseudoinverse defines these to be zero. One easy way to get the pseudoinverse, $H^{\dagger}$ is $H^{\dagger} = \frac{H^*}{H^*H}$

Note that if $H(u,v)$ is small then its inverse will be large, and so $H^{-1}(u,v)N(u,v)$ will be large. This is the source of the problem in the real system.

7.2 Wiener Filtering

Now we need to address the problems of inverse filtering. To do this, we need to define what we mean by "best", which means we need to define a way to measure how different things are. A common definition in engineering is the mean square error, MSE, for a stochastic variable

$$mse(x,\hat{x}) = E\left[(x-\hat{x})^2\right] \tag{7.10}$$

Let us say we wanted the estimate, $\hat{x}$ that minimized the MSE for an image, x, using a linear filter, $\hat{x} = g * y$. We further assume the noise is independent and identically distributed (iid), and the signal and noise are uncorrellated. Then in the Fourier space we have

$$\begin{aligned}
MSE(X,\hat{X}) &= E\left[(X-\hat{X})^2\right] && (7.11)\\
&= E\left[(X-GY)^2\right] && (7.12)\\
&= E\left[(X-G(HX+N))^2\right] && (7.13)\\
&= E\left[(X-GHX-GN)^2\right] && (7.14)\\
&= E\left[((I-GH)X-GN)^2\right] && (7.15)\\
&= E\left[((I-GH)X-GN)^*((I-GH)X-GN)\right] && (7.16)\\
&= E\left[(X^*(I-GH)^*-N^*G^*)((I-GH)X-GN)\right] && (7.17)\\
&= E\left[X^*(I-GH)^*((I-GH)X-GN)-N^*G^*((I-GH)X-GN)\right] && (7.18)\\
&= E\left[X^*(I-GH)^*(I-GH)X-X^*(I-GH)^*GN-N^*G^*(I-GH)X+N^*G^*GN\right] && (7.19)\\
&= E\left[X^*(I-GH)^*(I-GH)X+N^*G^*GN\right] && (7.20)\\
&= (I-GH)^*(I-GH)S_X+G^*GS_N && (7.21)\\
&= (I-H^*G^*)(I-GH)S_X+G^*GS_N && (7.22)\\
&= (I-H^*G^*-GH+H^*G^*GH)S_X+G^*GS_N && (7.23)
\end{aligned}$$

To minimize over G we need to take the gradient and set it equal to zero.

$$\begin{aligned}
\min_G MSE(X,\hat{X}) &= \nabla_G\left((I-H^*G^*-GH+H^*G^*GH)S_X+G^*GS_N\right) && (7.24)\\
0 &= -2H^*S_X+2GHH^*S_X+2GS_N && (7.25)\\
2H^*S_X &= 2G(HH^*S_X+S_N) && (7.26)\\
H^* &= G(HH^*+\frac{S_N}{S_X}) && (7.27)\\
G &= \frac{H^*}{(HH^*+\frac{S_N}{S_X})} && (7.28)
\end{aligned}$$

Table 7.1: Wiener filtering statistics.

Image	MSE	PSNR
Gaussian blur & Noise	1.4223e-04	38.4701
Wiener Filter	2.8775e-05	45.4099

Thus the best filter to invert a linear distortion in the minimum mean square error sense, assuming signal and noise are uncorrelated, and the noise is iid, is G,

$$G = \frac{H^*}{(HH^* + \frac{S_N}{S_X})} \tag{7.29}$$

Note three things. First, $HH^* = S_H$, i.e. the spectrum of the distortion, and $\frac{S_X}{S_N}$ is the SNR. You are thus turning down the gain on the system if the distortion is strong or the SNR is low. Second, if the noise goes to zero, then $S_N = 0$ and G becomes the pseudoinverse of H. This means in the no noise case the inverse filter is great, which we knew. Finally, in the noise case, the denominator of G grows (sum of positive numbers), so G decreases. When there is noise we thus turn down the gain on the filter so the noise does not get amplified to badly, and the gain setting is optimal in the mse sense.

7.3 Implementation

We need to supply the distortion model, H, and the spectrum of the noise, S_N, and the spectrum of the signal, S_X. Of these H and S_N can be either measured or modeled, and the filter is fairly immune to errors in S_X, so we are in fairly good shape.

Listing 7.1: MatLab Code For a Wiener Filter

```
function Iw=wiener(I,H,SN,SX)
    Hstar=conj(H);
    G=Hstar./(Hstar.*H+SN./SX);
    Iw=mat2gray(ifft2(G.*fft2(I)));
```

The results on a test image with both text (small with sharp edges) and facial features (large with soft edges) is quite good. The original image was blurred with a 5x5 gaussian blur with $\sigma = 0.5$ and gaussian noise with zero mean and standard deviation of .02. The Wiener filter cut the MSE by almost 80% (79.7685% to be more precise) and the Peak Signal to noise ratio improved by almost 7 decibels (6.9397db). Note the PSNR does not have to improve though it often will, as we were optimizing the MSE. The MSE thus must improve, but other measures like PSNR might or might not (it also depends on the maximum pixel value). The MSE and PSNR for both images is summarized below.

Figure 7.1: Wiener filtering Images.

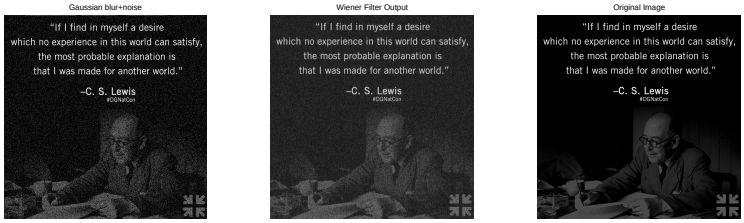


Figure 7.2: Zoomed view of Wiener filtering Images.

7.4 Constrained Least Squares

Let g be the measured image, h the distortion, f the undistorted image, and η a measure of the uncertainty. Thus our ideal of what the picture should be, $h * f$ should be within $\|\eta\|_2$ of g. This means

$$\|g - Hf\|_2 \quad = \quad \|\eta\|_2 \tag{7.30}$$

where H is the matrix representation of $h*$. We want this to be satisfied for some choice of η. In reality, there are usually many such f that work, so we need a requirement to pick which is best. Many options will, do. One popular one is to find the smoothest solution that meets the constraint, thus we would minimize the Laplacian of f. Other methods exist but we will choose this for now. The estimate of f is thus

$$\hat{f} \quad = \quad \text{argmin}_{\,f} \|Cf\|_2^2 \tag{7.31}$$

$$s.t.\ \|g - Hf\|_2^2 = \|\eta\|^2 \tag{7.32}$$

Chapter 8

Computed Tomography

How do you see inside an object? This question is of great importance to many areas such as medicine and non-destructive testing. This is the idea behind computed tomography. To achieve this goal we pass something, such as sound, magnetic fields, photons, ions, etc., through an object and measure what happened to the thing we passed through the object we want to know. The thing passed through the object is changed in some way by the object, and the measurements of how it is changed allow us to deduce what it passed through. Doing this repeatedly from different directions allow us to predict more and more of the object we want to know. I have stated this very generally because sound is attenuated by passing through objects, and the amount of the attenuation is what tells us what it passed through. Photons are absorbed and the amount absorbed tells us of the object. Ions lose energy through coulomb interactions, and this can tell us of the object. Magnetic fields can be stored and released, and so on. The underlying idea is the same, though some of the details change. I will attempt to keep the discussion general, though there are some important cases (parallel beam, cone beam, etc.), which will require a specific discussion to give you a proper understanding of the field. Without loss of generality we will consider only 2D images, as 3D computed Tomography images are just a series of stacked planes, see Figure 8.1.

8.1 Projections, Radon Transform, and the Sinogram

For this section we will assume the path taken by whatever you pass through the object to measure it, to be a straight line path. This is reasonable for many things, most notably xray CT. Realize some things, like ions do not follow straight line paths so a non-linear path must be used. This adds complexity, but does not change the underlying idea.

For the moment, consider a plane of detectors that is at some angle θ to the horizontal, see Figure 8.2. Now consider just one measurement taken, comprised of a single projection that is some distance R from a parallel projection line that passes through the origin. Note the distance R can easily be measured by the detector number. Since the projections are

Figure 8.1: The 3D CT image is composed of a series of stacked cut planes.

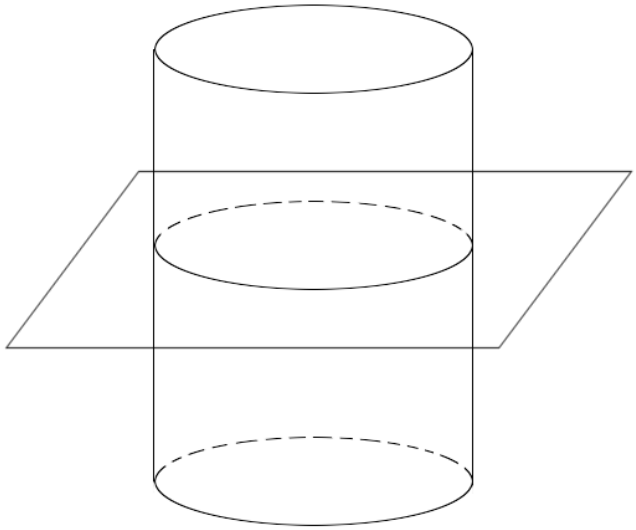


Figure 8.2: The Radon transform deals with a detector plane at angle θ to the horizontal, and a projection at some distance R from a parallel projection that passes through the origin of the system.

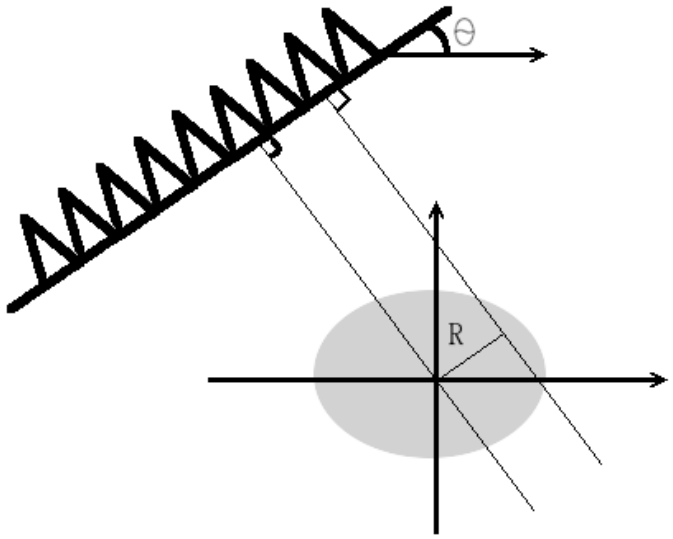
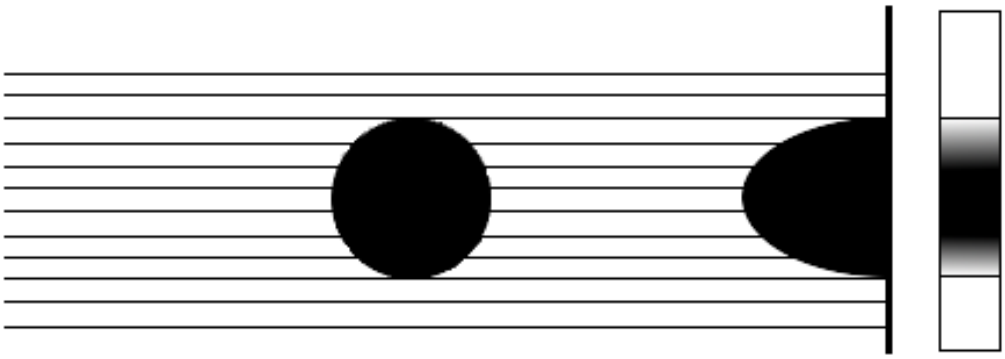


Figure 8.3: One object projected onto a plane, showing the measured attenuation/losses. These could be energy loses, photon loses, etc.



coming in orthogonal to the detectors and the detectors are measured counter-clock-wise from the horizontal, the

8.2 Parallel Beam CT

Parallel beam CT is exactly what its name suggests. A number of parallel measurements are taken by passing something (photons, sound, etc.) through the object, then measuring the loss or attenuation of the signal. Since each beam is a straight parallel line, see Figure 8.3, it is easy to know the path, and thus easier to reconstruct the image.

Figure 8.4: One object projected onto a series of planes in a CT scan.

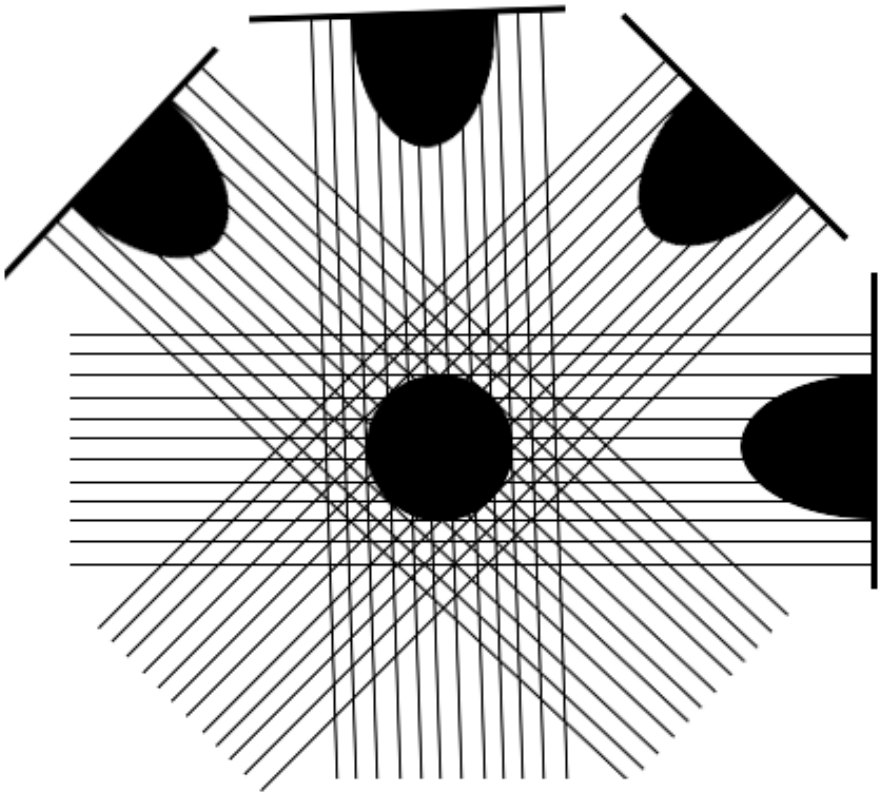


Figure 8.5: Two objects projected onto a series of planes in a CT scan.

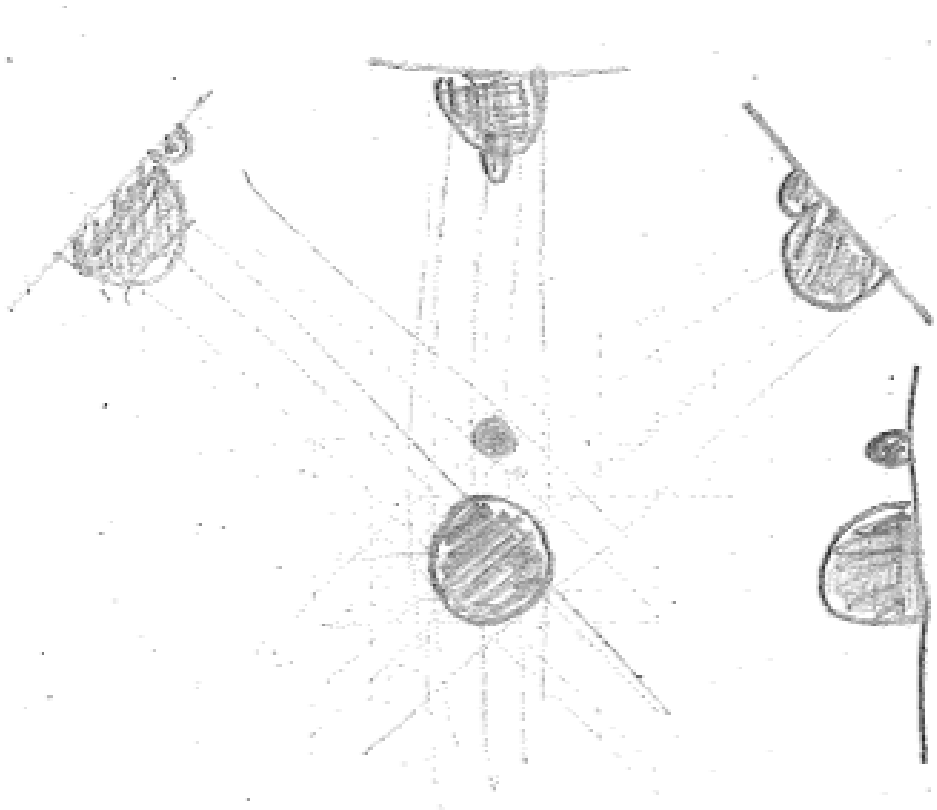


Figure 8.6: Sinogram for the two objects in Figure 8.5.

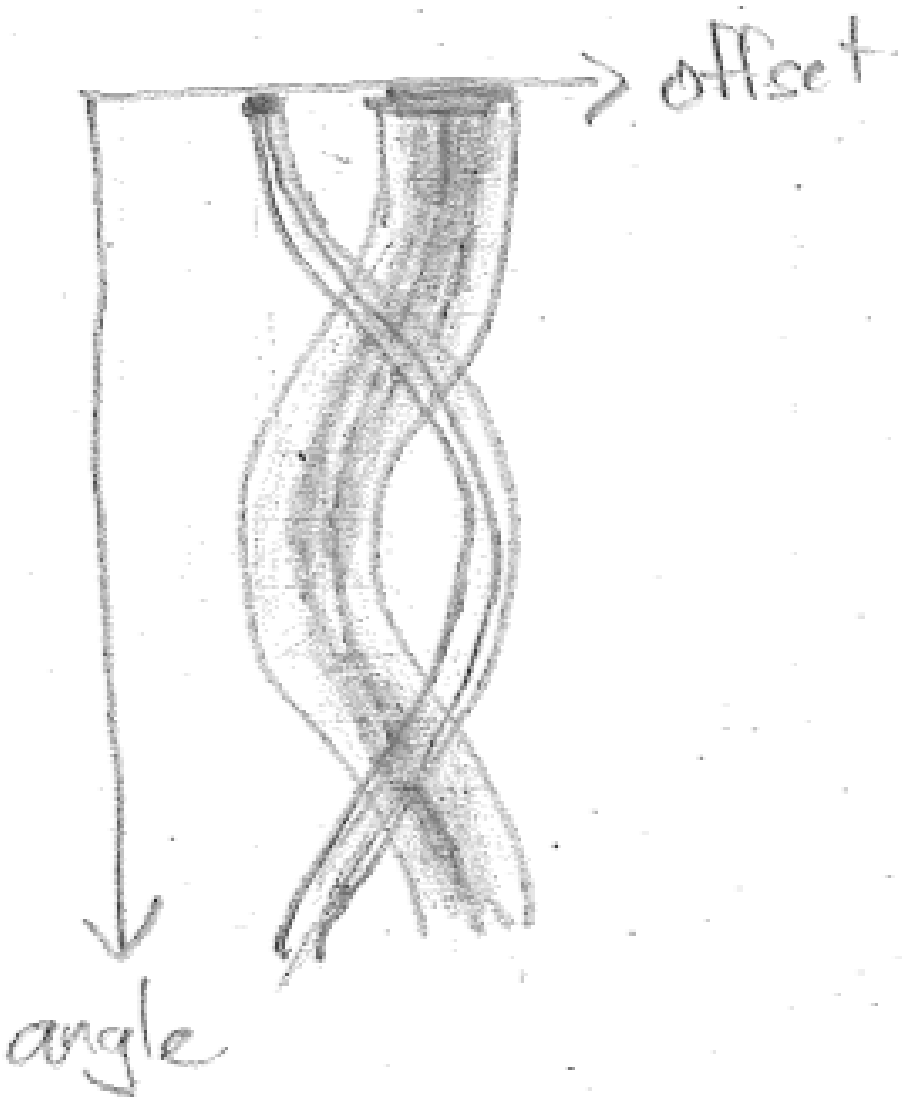
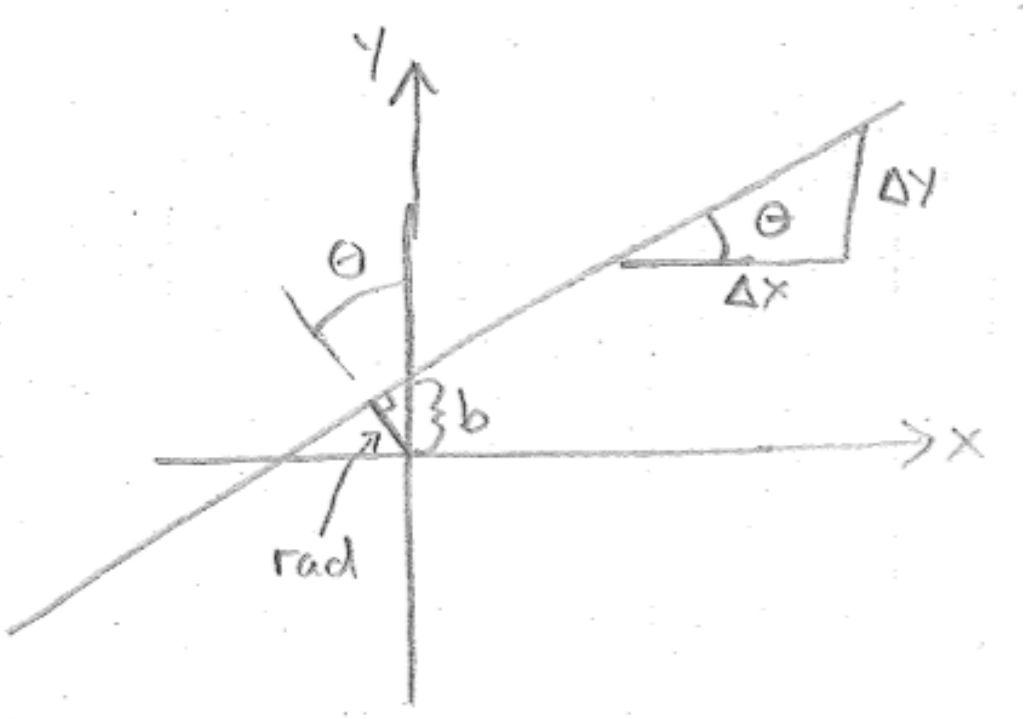


Figure 8.7: Projection line at rotational angle θ.



8.3 Fan Beam CT

8.4 Back Projection

8.5 Filtered Back Projection

8.6 Algebraic Reconstruction Technique

$$x_{element}(k+1) = x_{element}(k) + \frac{A_{projection,element} r_{projection}(k)}{A_{projection,*}} \tag{8.1}$$

$$A_{row,*} = \sum_{col=1}^{no\ of\ cols} A_{row,col} \tag{8.2}$$

$$r_{row}(k) = b_{row} - \sum_{col=1}^{no\ of\ cols} A_(row,col) x_{col}(k) \tag{8.3}$$

$$\tag{8.4}$$

Note that I wrote the first line using the names *projection* and *element*, but I used *row* and *col* on succeeding lines. This is because we update one projection at a time, but the projections are the rows of the A matrix so I wanted both the concepts to come across. Given name scoping rules, which I presume are familiar to the reader, I felt this was understandable. I would not recommend using straight ART, as SART has better convergence properties. I included it for completeness.

8.7 Simultaneous Algebraic Reconstruction Technique

$$x_{element}(k+1) = x_{element}(k) + \frac{\omega}{A_{*,element}} \sum_{row=1}^{no\ of\ rows} \frac{A_{row,element} r_{row}(k)}{A_{row,*}} \tag{8.5}$$

$$A_{*,element} = \sum_{row=1}^{no\ of\ rows} A_{row,element} \tag{8.6}$$

$$A_{row,*} = \sum_{col=1}^{no\ of\ cols} A_{row,col} \tag{8.7}$$

$$r_{row}(k) = b_{row} - \sum_{col=1}^{no\ of\ cols} A_(row,col) x_{col}(k) \tag{8.8}$$

$$\omega \in [0,2] \tag{8.9}$$

For the basic version of SART $\omega = 1$, though this relaxation parameter can be chosen to speed convergence. Note that these calculations can be done efficiently in sparse matrix formulation, which is usually the case.

8.8 Bundled Ordered Reconstruction of Images on Numerical GPGPU

Divide the x and b such that they are as non-overlapping as possible, this sets a blocking on the A matrix. The result is a series of problems, say indexed by i, such that $A_{i,j}x_j = b_i$. Note that due to sparsity and the mostly non-overlapping ordering, we have a correllation between i and j, alternately said the x_j are not necessarily independent, but they are close (similar for b_i). We then send each of these problems to a separate machine or gpgpu and do sub-rounds on each of the sub-problems, then after so many sub-iterations, we can update each other (either rotating/passing, partially, or fully) only with the portion of the data being worked on x_j. The result should converge to the actual solution. Bundles could even be split again into sub-bundles (same process) to handle clusters with multiple gpgpus. Should be very fast and is well tuned to the hardware.

Potential Issues:

1. Finding bundles so there is
   (a) some overlap (good updates),
   (b) not too much overlap (data transfer)
   (c) each bundle is not ill-conditioned
   (d) each bundle is not slow
2. Proof of convergence (reasonable given contraction mappings)
   (a) possibility of ill conditioned bundles above
   (b) possibility of updates causing instability by exciting one mode.
3. Does the image look good? (graininess, blurriness, etc.)

8.9 Proton/Ion CT Problem Setup

We will assume we want to reconstruct the contents, f_i for i the grid number of the region under consideration, see Fig. 8.8. We will do this by sending a particle[1] numbered k for $k \in \{1, 2, \cdots, n\}$, through the region and measuring their energy loss, E_k, entry angle, $\theta_{1,k}$, and exit angle, $\theta_{2,k}$. From the angles we calculate the most likely path (MLP). Note the MLP requires knowledge of the contents of the region, a "cathch-22" situation. We will

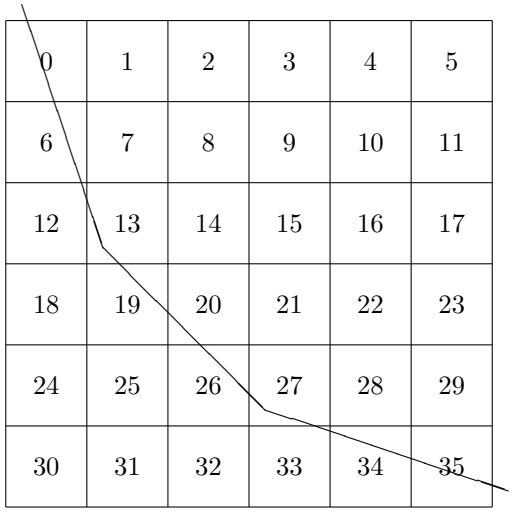


Figure 8.8: Trajectory of first particle.

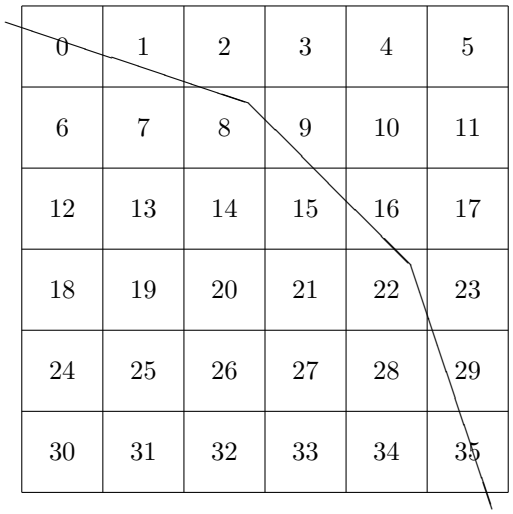


Figure 8.9: Trajectory of second particle.

overcome this by iterating, i.e.: assume a uniform contents, calculate the next iteration, then redo until it converges. Let's consider the trajectory of a particle.

In Fig. 8.8, we see the first particle's trajectory, which defines the first equation to be solved. Any square entered, even a little, by the particle will receive a weight of 1, otherwise the weight is 0. We will eventually make the weight proportional to the length of the trajectory in the square, but this makes the result easier to calculate now. The equation is

$$f_0 + f_6 + f_{12} + f_{13} + f_{19} + f_{20} + f_{26} + f_{27} + f_{33} + f_{34} + f_{35} \quad = \quad E_1.$$

Now consider the second particle's trajectory given in Fig. 8.9. The equation of this particle is

$$f_0 + f_1 + f_2 + f_8 + f_9 + f_{15} + f_{16} + f_{22} + f_{23} + f_{29} + f_{35} \quad = \quad E_2.$$

This is repeated for each particle, with the result in matrix notation, given by

$$WF \quad = \quad E$$

[1]This can also be done with waves, but the extension is trivial. My current research involves particles so I will use this case in the write-up.

where

$$W = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \cdots & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & \cdots & 0 & 0 & 0 & 1 \\ & & & & & & & & & \vdots & & & & \end{bmatrix}$$

$$F = \begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_{35} \end{bmatrix}$$

$$E = \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_n \end{bmatrix}$$

This could be solved for by any inversion technique but for large, sparse matrices this is usually solved for by iterative techniques, such as ART or SART.

Chapter 9

Removing Distortion From MR Images

9.1 Fitting a Plane

The equation of a 2 dimensional plane in 3 dimensional space is

$$\begin{aligned} Ax + By + Cz + D &= 0 \\ \begin{bmatrix} A \\ B \\ C \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + D &= 0. \end{aligned}$$

To fit this to a bunch of data we need to separate out one of the dimensions that we know is not orthogonal (or near orthogonal) to the normal from the plane. If you don't do this, then the system has the degenerate solution $A = B = C = D = 0$.

$$\begin{aligned} Cz &= -Ax - By - D \\ z &= -\frac{A}{C}x - \frac{B}{C}y - \frac{D}{C} \\ &= ax + by + d \end{aligned}$$

Now we proceed by noting that we want to fit the plane to a bunch of points (x_i, y_i, z_i) so we have one equation for each unknown.

$$\begin{aligned}
\begin{bmatrix} z_0 \\ z_1 \\ \vdots \\ z_{n-1} \end{bmatrix} &= \begin{bmatrix} ax_0 + by_0 + d \\ ax_1 + by_1 + d \\ \vdots \\ ax_{n-1} + by_{n-1} + d \end{bmatrix} \\
&= \begin{bmatrix} x_0 & y_0 & 1 \\ x_0 & y_0 & 1 \\ \vdots & \vdots & \vdots \\ x_{n-1} & y_{n-1} & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ d \end{bmatrix}
\end{aligned}$$

This equation can easily be solved for the values a, b, and d using LU, QR, SVD, or your favorite technique.

The norm, N, is then

$$\begin{aligned}
n &= \begin{bmatrix} -a \\ -b \\ 1 \end{bmatrix} \\
N &= \frac{n}{\|n\|} \\
&= \begin{bmatrix} -\frac{a}{\sqrt{1+a^2+b^2}} \\ -\frac{b}{\sqrt{1+a^2+b^2}} \\ \frac{1}{\sqrt{1+a^2+b^2}} \end{bmatrix}.
\end{aligned}$$

For a point, p, the equation of the plane is thus

$$\begin{aligned}
p^T N &= \frac{d}{\|n\|} \\
p^T N &= d_1.
\end{aligned}$$

9.2 Moving a Plane

Let a fitted plane be defined by

$$p^T N = d_1.$$

To move a plane a distance h in the direction of the normal vector we have in essence, that each point (P_1) is moved by hN to a new point (P_2).

$$P_2 = P_1 + hN \tag{9.1}$$

Also the new, shifted plane must have the same normal (or it wouldn't be parallel).

$$p^T N = d_2 \tag{9.2}$$

Using both Eq 9.1 and Eq 9.2 we obtain

$$\begin{aligned} d_2 &= P_2^T N \\ &= P_1^T N + hN^T N \\ &= d_1 + h \\ &= \frac{d}{\sqrt{1+a^2+b^2}} + h. \end{aligned}$$

9.3 Undistorted Data

For an acquired point (x, y, z) with a corresponding undistorted point ($\bar{x}$, $\bar{y}$, $\bar{z}$) we shall assume that the plane is essentially in the x,y plane, hence we will let $x = \bar{x}$ and $y = \bar{y}$ be the grid for the plane. Using this and the equation of the plane from above can be used to find $\bar{z}$.

$$\bar{z} = ax + by + d$$

Bibliography