Keith On . . .
Robotics and Control

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

List of Tables

Part I

Robotics

Chapter 1

Forward Kinematics

1.1 A Very Simple Robot

Let's start our mathematical modeling with a very simple robot that has only one radial joint at the origin, with an arm of length l (see Figure 1.1). We can easily find that the location of the point (p_x, p_y) is given by trigonometry.

$$p_x = l\cos(\theta) \tag{1.1}$$
$$p_y = l\sin(\theta) \tag{1.2}$$

We could stop here, but let's put this in matrix notation. First we need to agree on a convention for writing the matrices. For a two dimensional problem we need to know the orientation and location. We will assume you are always facing in the direction of your local x coordinate, which I will call the Q direction. We will specify your orientation by giving your local Q and R axis in the coordinate system you are in (say x and y).

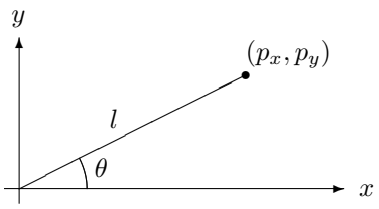
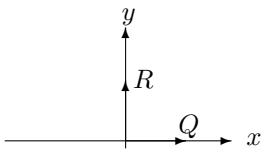
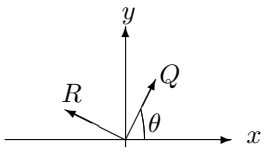
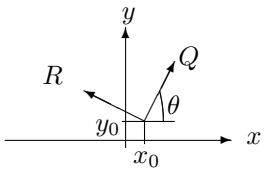


Figure 1.1: Single Radial Arm

$$\begin{bmatrix} \begin{bmatrix} Q_x & R_x \\ Q_y & R_y \end{bmatrix} & \begin{bmatrix} x_{QR} \\ y_{QR} \end{bmatrix} \\ \begin{bmatrix} 0 & 0 \end{bmatrix} & 1 \end{bmatrix}$$

Note that the upper left sub-matrix is the orientation block, the upper right sub-matrix is the position block of the origin of your local coordinate, and the lower sub-matrix is a format to make multiplication easy. Lets look at some examples to try to make this clear

$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	
$\begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$	
$\begin{bmatrix} \cos(\theta) & -\sin(\theta) & x_0 \\ \sin(\theta) & \cos(\theta) & y_0 \\ 0 & 0 & 1 \end{bmatrix}$	

Thus we can write the position of the dot in Figure 1.1 as a rotation of the local coordinates by θ followed by a travel of l in the new local x direction.

$$R(\theta)T(l) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & l \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & l\cos(\theta) \\ \sin(\theta) & \cos(\theta) & l\sin(\theta) \\ 0 & 0 & 1 \end{bmatrix}$$

Note the position block of the product is the location of the dot from Figure 1.1. As an added advantage we also have the local coordinates. This probably seems like a lot of work for a little benefit, but the real advantage can be seen in the next section, when we deal with a multi-segment robot.

1.2 A Two Segment Robot

Now we will consider a robotic arm with two radial joints and two arm segments (see Figure 1.2). You could do the geometric calculations by summing the x and y components

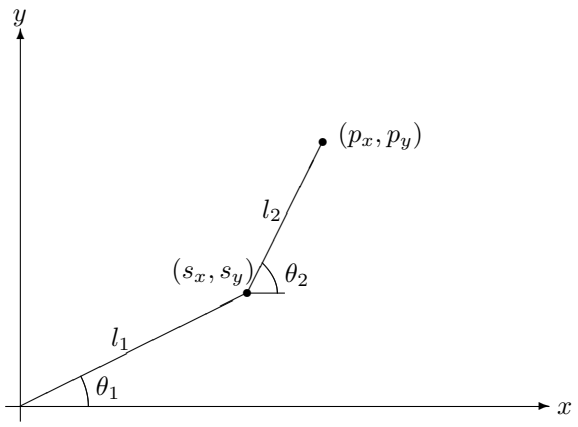


Figure 1.2: Arm With Two Radial Joints

of the two arms.

$$\begin{aligned} p_x &= s_x + l_2\cos(\theta_2) \\ &= l_1\cos(\theta_1) + l_2\cos(\theta_2) \\ p_y &= s_y + l_2\sin(\theta_2) \\ &= l_1\sin(\theta_1) + l_2\sin(\theta_2) \end{aligned}$$

This is quite straightforward. Now consider our matrix methods to find the location, L.

$$\begin{aligned} L &= R(\theta_1)T(l_1)R(\theta_2-\theta_1)T(l_2) \\ &= \begin{bmatrix} \cos(\theta_1) & -\sin(\theta_1) & 0 \\ \sin(\theta_1) & \cos(\theta_1) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & l_1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\theta_2-\theta_1) & -\sin(\theta_2-\theta_1) & 0 \\ \sin(\theta_2-\theta_1) & \cos(\theta_2-\theta_1) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & l_2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} \cos(\theta_1) & -\sin(\theta_1) & l_1\cos(\theta_1) \\ \sin(\theta_1) & \cos(\theta_1) & l_1\sin(\theta_1) \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\theta_2-\theta_1) & -\sin(\theta_2-\theta_1) & l_2\cos(\theta_2-\theta_1) \\ \sin(\theta_2-\theta_1) & \cos(\theta_2-\theta_1) & l_2\sin(\theta_2-\theta_1) \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} a & -b & l_1\cos(\theta_1) + l_2 a \\ b & a & l_1\sin(\theta_1) + l_2 b \\ 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

where,

$$\begin{aligned} a &= \cos(\theta_1)\cos(\theta_2 - \theta_1) - \sin(\theta_1)\sin(\theta_2 - \theta_1) \\ &= \cos(\theta_2) \\ b &= \sin(\theta_1)\cos(\theta_2 - \theta_1) + \sin(\theta_1)\cos(\theta_2 - \theta_1) \\ &= \sin(\theta_2) \end{aligned}$$

thus we have,

$$L = \begin{bmatrix} \cos(\theta_2) & -\sin(\theta_2) & l_1\cos(\theta_1) + l_2\cos(\theta_2) \\ \sin(\theta_2) & \cos(\theta_2) & l_1\sin(\theta_1) + l_2\sin(\theta_2) \\ 0 & 0 & 1 \end{bmatrix}.$$

This hardly seems as easy, so why do it? Quite honestly because

- it is straightforward to program,
- since all angles will actually be measured relatively[1]some of the ugliness of the matrix form goes away and some of the niceness of the summation also disappears,
- in higher dimensions with many actuators (rotational, prismatic, and spherical), it can be confusing for the summation form, but the matrix form is still easy to write the equation and the computation grows m^3n for m the number of dimensions and n the number of segments.

[1]Thus we really measure $\theta_2 - \theta_1$ not θ_2.

Chapter 2

Inverse Kinematics

Forward kinematics is the calculation of the position and orientation of a manipulator, knowing the values of the variables that describe the robotic system. Inverse kinematics is the reverse of this; namely find the values of the variables that describe the robotic system that will make it reach a given position and orientation. Ideally we would have a closed form solution, which would then allow us to simply plug in the numbers and calculate the answer, quickly and directly. A closed form solution is not always possible so other techniques are also available.

2.1 Closed Form Techniques

2.1.1 Algebraic

2.1.2 Geometric

2.1.3 Separable

2.2 Numerical Techniques

2.2.1 Zero Finding

Forward kinematics equations are simple and direct to obtain. Let the forward kinematics of a robotic system be given by

$$Pos \quad = \quad f(Params) \tag{2.1}$$

Where $Params$ is a vector of the unknown variables that describe the state of the robotic system (usually a collection of lengths and angles), and Pos is the vector which describes the current position (possibly a vectorized version of the position matrix from last chapter). We note that Pos is known and that we can rewrite this as $g(Params) = f(Params) - Pos = 0$, thus if we find when $g(\cdot)$ is zero, then we have solved the inverse kinematics problem.

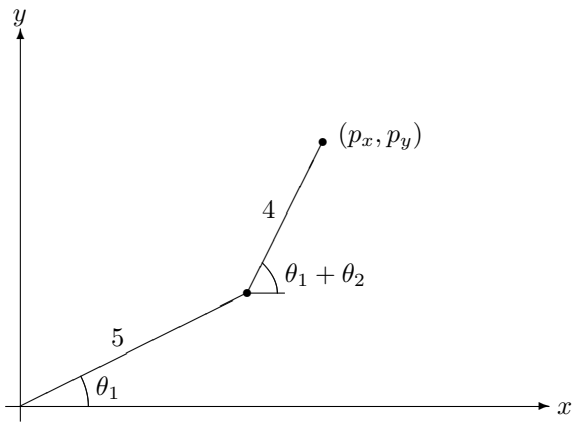


Figure 2.1: Arm With Two Radial Joints

Finding out when a function goes to zero is a standard numerics problem, and the reader is referred to the chapter on solving these problems in KONA. Since selecting a method can be challenging, the reader is encouraged to consider Newton's method (fast, though more computationally complex, and potentially unstable) and bisection (slow, but simple and safe).

2.2.2 Optimization

Similar to what was done in zero finding, we could also consider finding when $\|g(\cdot)\|$ is a minimum. This will happen when $g(\cdot)$ is zero, as before, but it can be calculated a different way.

Example 1 *Consider solving the inverse kinematics problem for the robot in Figure 2.1 when the wrist is at the point* $p = [5 \quad 6]^T$.

Solution:

First, we note the forward kinematics solution is given by

$$\begin{aligned} p_x &= 5\cos(\theta_1) + 4\cos(\theta_1 + \theta_2) \\ p_y &= 5\sin(\theta_1) + 4\sin(\theta_1 + \theta_2), \end{aligned}$$

and the two norm of this is the cost we will minimize is

$$\begin{aligned} cost &= (p_x - 5)^2 + (p_y - 6)^2 \\ &= (5\cos(\theta_1) + 4\cos(\theta_1 + \theta_2) - 5)^2 + (5\sin(\theta_1) + 4\sin(\theta_1 + \theta_2) - 6)^2. \end{aligned}$$

The gradient of this with respect to θ_i is thus

$$\begin{aligned} \nabla_\theta cost &= \begin{bmatrix} 2(p_x - 5)\frac{\partial}{\partial\theta_1}p_x + 2(p_y - 6)\frac{\partial}{\partial\theta_1}p_y \\ 2(p_x - 5)\frac{\partial}{\partial\theta_2}p_x + 2(p_y - 6)\frac{\partial}{\partial\theta_2}p_y \end{bmatrix} \\ &= \begin{bmatrix} 2(5c_1 + 4c_{1,2} - 5)(-5s_1 - 4s_{1,2}) + 2(5s_1 + 4s_{1,2} - 6)(5c_1 + 4c_{1,2}) \\ 2(5c_1 + 4c_{1,2} - 5)(-4s_{1,2})) + 2(5s_1 + 4s_{1,2} - 6)(4c_{1,2}) \end{bmatrix}. \end{aligned}$$

Note, to make this fit, I have used the common notation of $c_i = \cos(\theta_i)$ and $s_i = \sin(\theta_i)$. Commas in the subscript denote multiple angles are added together.

To solve this numerically, I will use SciLab's optim *function. In MatLab, I could use* fminu*. Consider the code in Listing 2.1, which defines both cost and $\nabla_\theta cost$ for our robot (ind is used for debugging and is ignored here, but must be included). Essentially this function tells us how well we did (cost) and the direction of a better solution ($\nabla_\theta cost$). We only need to supply an initial guess then repeatedly call the function, while adjusting the guess given the gradient. This is what the function optim does. The call to optim is shown in Listing 2.2, and it produces the output below. Note angles are in radians.*

```
theta  =

    0.4165215
    1.0471976

 p  =

    5.
    6.
```

Listing 2.1: Cost function to optimize.

```
function [cost,grad,ind]=invKin(x,ind)
  c1=cos(x(1));
  c12=cos(x(1)+x(2));
  s1=sin(x(1));
  s12=sin(x(1)+x(2));
  term1=5*c1+4*c12-5;
  d1term1=-5*s1-4*s12;
  d2term1=-4*s12;
  term2=5*s1+4*s12-6;
  d1term2=5*c1+4*c12;
```

```
  d2term2=4*c12;
  cost=term1^2+term2^2;
  grad=[term1*d1term1+term2*d1term2
        term1*d2term1+term2*d2term2];
endfunction
```

Listing 2.2: Calling the optimization routine.

```
theta1=%pi/3;
theta2=%pi/3;
x0=[theta1
    theta2];
[cost,x]=optim(invKin,x0);

theta=x
p=[5*cos(x(1))+4*cos(x(1)+x(2))
   5*sin(x(1))+4*sin(x(1)+x(2))]
```

Chapter 3

Motion and Trajectories

3.1 Differentials

3.2 Jacobian

The Jacobian transforms differential changes in one set of variable to differential changes in another set of variables.

3.3 Trajectory Planning

Chapter 4

Mechanics

4.1 Euler-Lagrange

Consider the potential energy of a rigid object at height, y, in a gravitational field.

$$\mathfrak{P} = mgy \tag{4.1}$$

Notice that this is the force applied by gravity over the distance to bring the object to the ground. Thus, we see the derivative of the potential energy with respect to the variable representing the displacement of the object gives the force that the field applies.

$$\frac{\partial}{\partial y}\mathfrak{P} = mg \tag{4.2}$$

$$= F_g \tag{4.3}$$

Now consider the kinetic energy of a moving rigid body.

$$\mathfrak{K} = \frac{1}{2}mv^2 \tag{4.4}$$

The momentum is related to the kinetic energy by the derivative with respect to the velocity.

$$\frac{\partial}{\partial v}\mathfrak{K} = mv \tag{4.5}$$

$$= p \tag{4.6}$$

Now recall that the momentum and force are related by $F\Delta t = mv$ which means $F = (mv)/\Delta t$ or we should consider the time derivative of the momentum.

$$\frac{d}{dt}p = \frac{d}{dt}m\dot{v} \tag{4.7}$$

$$= m\ddot{v} \tag{4.8}$$

$$= F \tag{4.9}$$

We thus have expressions for everything we care about in terms of forces.

$$\begin{aligned} \mathfrak{L} &= \mathfrak{K} - \mathfrak{P} && (4.10)\\ \sum F &= \frac{d}{dt}\frac{\partial \mathfrak{L}}{\partial \dot{x}} - \frac{\partial \mathfrak{L}}{\partial x} && (4.11)\\ &= 0 && (4.12) \end{aligned}$$

The final equation (Eq. 4.12) assumes that no outside forces are acting on the system. Similarly, we can calculate the torques on a system.

$$\begin{aligned} \sum T &= \frac{d}{dt}\frac{\partial \mathfrak{L}}{\partial \dot{\theta}} - \frac{\partial \mathfrak{L}}{\partial \theta} && (4.13)\\ &= 0 && (4.14) \end{aligned}$$

Example 2 *Consider a single rotational joint robotic arm of length l and mass m, that rotates about one end ($I = \frac{1}{3}mL^2$), with angle θ measured from the negative "y-axis" to conform with typical physics developments. Calculate the dynamical equations of motion.*

Solution:

The kinetic energy, K, is entirely rotational[1]*, and thus:*

$$K = \frac{1}{2}I\dot{\theta}^2. \quad (4.24)$$

[1] If this does not make immediate sense, consider the following. $K = K_x + K_y$, i.e. break it into kinetic energy in the x and y directions.

$$\begin{aligned} K_x &= \frac{1}{2}\int_0^l \frac{m}{l} r^2 dr \dot{\theta}^2 sin^2(\theta) && (4.15)\\ &= \frac{1}{2}\frac{m}{l}\int_0^l r^2 dr \dot{\theta}^2 sin^2(\theta) && (4.16)\\ &= \frac{1}{2}\frac{m}{l}\left.\frac{r^3}{3}\right|_0^l \dot{\theta}^2 sin^2(\theta) && (4.17)\\ &= \frac{1}{2}\frac{m}{l}\left(\frac{l^3}{3}\right)\dot{\theta}^2 sin^2(\theta) && (4.18)\\ &= \frac{1}{2}m\left(\frac{l^2}{3}\right)\dot{\theta}^2 sin^2(\theta) && (4.19)\\ &= \frac{1}{2}I\dot{\theta}^2 sin^2(\theta) && (4.20) \end{aligned}$$

and similarly for K_y yields

$$\begin{aligned} K_y &= \frac{1}{2}\int_0^l \frac{m}{l} r^2 dr \dot{\theta}^2 cos^2(\theta) && (4.21)\\ &= \frac{1}{2}I\dot{\theta}^2 cos^2(\theta) && (4.22) \end{aligned}$$

and thus since $\sin^2(\theta) + \cos^2(\theta) = 1$,

$$K = \frac{1}{2}I\dot{\theta}^2 \quad (4.23)$$

The potential energy, P, is based off how high the center of the mass is, so it is:

$$P = mg\frac{l}{2}(1-\cos(\theta)). \tag{4.25}$$

The Lagrangian is the difference:

$$L = \frac{1}{2}I\dot{\theta}^2 - mg\frac{l}{2}(1-\cos(\theta)). \tag{4.26}$$

Now using Equation 4.14, we have

$$0 = \left(\frac{d}{dt}I\dot{\theta}\right) - \left(-mg\frac{l}{2}(\sin(\theta))\right) \tag{4.27}$$

$$= I\ddot{\theta} + mg\frac{l}{2}\sin(\theta). \tag{4.28}$$

Note that if the system is not in equilibrium (i.e. your motor is applying a torque τ to the system, then

$$\tau = I\ddot{\theta} + mg\frac{l}{2}\sin(\theta). \tag{4.29}$$

As an interesting extension, we could consider a motor with gear ratio g and moment of inertia I_m then the new dynamics would be

$$\tau = (I + g^2 I_m)\ddot{\theta} + mg\frac{l}{2}\sin(\theta). \tag{4.30}$$

If we wanted to add damping, C_b and C_m, then

$$\tau = (I + g^2 I_m)\ddot{\theta} + (C_b + gC_m)\dot{\theta} + mg\frac{l}{2}\sin(\theta). \tag{4.31}$$

Chapter 5

Control

5.1 Stability

5.2 Modeling

Imagine an inverted pendulum attached to a cart, which we can apply a force to, see Figure 5.2. We can analyze this using Lagrangian mechanics.

Energy Term	Value
Kinetic Energy of the Cart	$\frac{1}{2}m_1\dot{x}^2$
Kinetic Energy of the Pendulum	$\frac{1}{2}m_2(\dot{x} + l\cos(\theta)\dot{\theta})^2 + \frac{1}{2}m_2(l\sin(\theta)\dot{\theta})^2$
Potential Energy of the Pendulum	$m_2 gl\cos(\theta)$

To obtain the equation of linear motion in the pendulum cart system, we set the external force equal to the sum of forces in Eq. 4.11 on page 14 we obtain:

$$F = m_1\ddot{x} + m_2(\ddot{x} + l\cos(\theta)\dot{\theta}) \tag{5.1}$$

Then to obtain the equations of angular motion, we take the equilibrium condition of Eq. 4.13 on page 14 to obtain:

$$0 = m_2(\dot{x} + l\cos(\theta)\dot{\theta})(-l\sin(\theta)\dot{\theta}+) \tag{5.2}$$

5.3 Transforms

5.4 PID

5.5 Lead-Lag

Figure 5.1: Inverted Pendulum Cart.

Part II

Transforms

Chapter 6

Convolution

Transform techniques covers a wide variety of mathematical methods. Probably the most used are the Laplace transform, the Fourier transforms, and the Z-transform. To really cover these well I need to describe convolution, which is a fundamental operation many people have not heard of before. Convolution is probably best known as polynomial multiplication, but its application is far broader than this one case. Convolution describes how a system $h(t)$ responds to an input $g(t)$, to yield an output $h(t) * g(t)$. This is very general as the system could be air with smog or dust, and the input some pattern of light (picture), which then produces an output (a blurred picture). Without getting caught up on the beauty and usefulness of this we will look at a simple example.

$$\begin{aligned} C(x) &= A(x) * B(x) \\ &= (5x^3 - 7x^2 + 6x + 4)(2x^2 - x + 3) \\ &= (5 \cdot 2)x^5 + (5 \cdot (-1) + (-7) \cdot 2)x^4 + (5 \cdot 3 + (-7) \cdot (-1) + 6 \cdot 2)x^3 \\ &\quad +((-7) \cdot 3 + 6 \cdot (-1) + 4 \cdot 2)x^2 + (6 \cdot 3 + 4 \cdot (-1))x + (4 \cdot 3) \\ &= (10)x^5 + (-5 - 14)x^4 + (15 + 7 + 12)x^3 + (-21 - 6 + 8)x^2 + (18 - 4)x + (12) \\ &= 10x^5 - 19x^4 + 34x^3 - 19x^2 + 14x + 12 \end{aligned}$$

Alternately we could describe this in vector notation as $A = \begin{bmatrix} 5 & -7 & 6 & 4 \end{bmatrix}$ and $B = \begin{bmatrix} 2 & -1 & 3 \end{bmatrix}$. Convolution then corresponds to flipping one of the two vectors (it doesn't matter which) and then shifting one relative to the other. At each shift we multiply the overlapping terms and sum. Blank terms are equal to zero[1]. Each shift, k, gives us one of the coefficients, c_k. Note that k is the distance from the zero elements. An example is best.

B					2	-1	3
A	4	6	-7	5			
c_6	0	0	0	0	0	0	0

[1]This is just noting that you can express any missing monomial power, x^p as $0 \cdot x^p$, and thus add it to the polynomial expansion, yielding a '0' in the vector.

This tells us that the x^6 term does not appear in the product. To see that this is the $k = 6$ term note that it take 6 hops to go from the 4 in A to the 3 in B. Doing this for the (4+3-1) shifts (number of coefficients in both vectors minus one) we find the following.

B				2	-1	3	
A	4	6	-7	5			$c_5 = 10$
c_5	0	0	0	10	0	0	

B			2	-1	3	
A	4	6	-7	5		$c_4 = -19$
c_4	0	0	-14	-5	0	

B		2	-1	3	
A	4	6	-7	5	$c_3 = 34$
c_3	0	12	7	15	

B	2	-1	3		
A	4	6	-7	5	$c_2 = -19$
c_2	8	-6	-21	15	

B	2	-1	3			
A		4	6	-7	5	$c_1 = 14$
c_1	0	-4	18	0	0	

B	2	-1	3				
A			4	6	-7	5	$c_0 = 12$
c_0	0	0	12	0	0	0	

Trivially you can see this is the same as we found on the polynomial technique.

6.1 Images

For another use of convolution, I will consider a one dimensional "skyline" image. I do this to avoid the computation of the two dimensional version (it is messy to see as your first example).

6.2 Rapid Calculation

Now we consider the fast way to do this[2]. Note that most people would never consider doing this, but for large problems it involves a lot less work. We will consider the algorithm for

[2]This turns out to be take the fast fourier transform, multiply, and take the inverse fast fourier transform. The reason I do it this way is that I want you to know we can approach a problem from a lot of ways and still get the same truth. There is no magic, only clear thinking.

equal length vectors A and B, which can be achieved by zero padding the smaller. Assume the length (number of coefficients) is n.

1. Evaluate $A(x)$ and $B(x)$ at $2n$ points, x_i.

2. Calculate $C(x_i) = A(x_i)B(x_i)$.

3. Evaluate $D(x) = \sum_{i=0}^{2n-1} C(x_i)x^i$ at $2n$ points, x_k.

4. Calculate $c_k = \frac{D(x_k)}{2n}$

The evaluation steps must be done in $O(n\log(n))$ steps, which we will do by a divide and conquer method, namely we find that $A(x) = A_{even}(x^2) + xA_{odd}(x^2)$. We now pick our $x_k = e^{\frac{j\pi}{n}k}$, where $j = \sqrt{-1}$. Note I am an engineer and we don't use i for the imaginary number because i is current. Let's do the algorithm for our test problem.

First, we find $A(x_i)$ and $B(x_i)$, noting that $n = 4$, so $x_i = e^{\frac{j\pi}{4}k}$ and

$$\begin{aligned} X &= \begin{bmatrix} e^{\frac{j\pi}{4}0} & e^{\frac{j\pi}{4}1} & e^{\frac{j\pi}{4}2} & e^{\frac{j\pi}{4}3} & e^{\frac{j\pi}{4}4} & e^{\frac{j\pi}{4}5} & e^{\frac{j\pi}{4}6} & e^{\frac{j\pi}{4}7} \end{bmatrix} \\ &= \begin{bmatrix} e^{0} & e^{\frac{j\pi}{4}} & e^{\frac{j\pi}{2}} & e^{\frac{j\pi}{4}3} & e^{j\pi} & e^{\frac{j\pi}{4}+j\pi} & e^{\frac{j\pi}{2}+j\pi} & e^{\frac{j\pi}{4}3+j\pi} \end{bmatrix} \\ &= \begin{bmatrix} 1 & e^{\frac{j\pi}{4}} & e^{\frac{j\pi}{2}} & e^{\frac{j3\pi}{4}} & -1 & -e^{\frac{j\pi}{4}} & -e^{\frac{j\pi}{2}} & -e^{\frac{j3\pi}{4}} \end{bmatrix} \end{aligned}$$

and thus

$$\begin{aligned} X^2 &= \begin{bmatrix} 1^2 & e^{\frac{j\pi}{4}2} & e^{\frac{j\pi}{2}2} & e^{\frac{j3\pi}{4}2} & (-1)^2 & (-e^{\frac{j\pi}{4}})^2 & (-e^{\frac{j\pi}{2}})^2 & (-e^{\frac{j3\pi}{4}})^2 \end{bmatrix} \\ &= \begin{bmatrix} 1 & e^{\frac{j\pi}{2}} & e^{j\pi} & e^{\frac{j\pi}{2}+j\pi} & 1 & e^{\frac{j\pi}{2}} & e^{j\pi} & e^{\frac{j\pi}{2}+j\pi} \end{bmatrix} \\ &= \begin{bmatrix} 1 & e^{\frac{j\pi}{2}} & -1 & -e^{\frac{j\pi}{2}} & 1 & e^{\frac{j\pi}{2}} & -1 & -e^{\frac{j\pi}{2}} \end{bmatrix}. \end{aligned}$$

Notice that the first half of X^2 is identical to the second half, so only one must be calculated. This observation is needed to obtain the $O(n\log(n))$ complexity. I will use ".*" to mean

array (element by element) multiplication. So $A(X)$ is

$$
\begin{aligned}
A(X) &= (5X^3 - 7X^2 + 6X + 4) \\
&= (-7X^2 + 4) + X. * (5X^2 + 6) \\
&= \begin{bmatrix} 4-7*1 \\ 4-7e^{\frac{j\pi}{2}} \\ 4-7(-1) \\ 4-7(-e^{\frac{j\pi}{2}}) \\ 4-7*1 \\ 4-7e^{\frac{j\pi}{2}} \\ 4-7(-1) \\ 4-7(-e^{\frac{j\pi}{2}}) \end{bmatrix} + \begin{bmatrix} 1 \\ e^{\frac{j\pi}{4}} \\ e^{\frac{j\pi}{2}} \\ e^{\frac{j3\pi}{4}} \\ -1 \\ -e^{\frac{j\pi}{4}} \\ -e^{\frac{j\pi}{2}} \\ -e^{\frac{j3\pi}{4}} \end{bmatrix} . * \begin{bmatrix} 6+5*1 \\ 6+5*e^{\frac{j\pi}{2}} \\ 6+5*(-1) \\ 6+5*(-e^{\frac{j\pi}{2}}) \\ 6+5*1 \\ 6+5*e^{\frac{j\pi}{2}} \\ 6+5*(-1) \\ 6+5*(-e^{\frac{j\pi}{2}}) \end{bmatrix} \\
&= \begin{bmatrix} -3 \\ 4-7j \\ 11 \\ 4+7j \\ -3 \\ 4-7j \\ 11 \\ 4+7j \end{bmatrix} + \begin{bmatrix} 1 \\ e^{\frac{j\pi}{4}} \\ e^{\frac{j\pi}{2}} \\ e^{\frac{j3\pi}{4}} \\ -1 \\ -e^{\frac{j\pi}{4}} \\ -e^{\frac{j\pi}{2}} \\ -e^{\frac{j3\pi}{4}} \end{bmatrix} . * \begin{bmatrix} 11 \\ 6+5j \\ 1 \\ 6-5j \\ 11 \\ 6+5j \\ 1 \\ 6-5j \end{bmatrix} \\
&= \begin{bmatrix} -3 \\ 4-7j \\ 11 \\ 4+7j \\ -3 \\ 4-7j \\ 11 \\ 4+7j \end{bmatrix} + \begin{bmatrix} 11 \\ 6e^{\frac{j\pi}{4}} + 5e^{\frac{j3\pi}{4}} \\ j \\ 6e^{\frac{j3\pi}{4}} + 5e^{\frac{j\pi}{4}} \\ -11 \\ -6e^{\frac{j\pi}{4}} - 5e^{\frac{j3\pi}{4}} \\ -j \\ -6e^{\frac{j3\pi}{4}} - 5e^{\frac{j\pi}{4}} \end{bmatrix} \\
&= \begin{bmatrix} 8 \\ 4-7j+6e^{\frac{j\pi}{4}} + 5e^{\frac{j3\pi}{4}} \\ 11+j \\ 4+7j+6e^{\frac{j3\pi}{4}} + 5e^{\frac{j\pi}{4}} \\ -14 \\ 4-7j-6e^{\frac{j\pi}{4}} - 5e^{\frac{j3\pi}{4}} \\ 11-j \\ 4+7j-6e^{\frac{j3\pi}{4}} - 5e^{\frac{j\pi}{4}} \end{bmatrix} \\
&= \begin{bmatrix} 8 \\ 4+\frac{1}{\sqrt{2}} + (\frac{11}{\sqrt{2}} - 7)j \\ 11+j \\ 4-\frac{1}{\sqrt{2}} + (\frac{11}{\sqrt{2}} + 7)j \\ -14 \\ 4-\frac{1}{\sqrt{2}} - (\frac{11}{\sqrt{2}} + 7)j \\ 11-j \\ 4+\frac{1}{\sqrt{2}} - (\frac{11}{\sqrt{2}} - 7)j \end{bmatrix},
\end{aligned}
$$

and $B(X)$ is

$$\begin{aligned}
B(X) &= 2X^2 - X + 3 \\
&= (2X^2 + 3) + X. * (0X^2 + 1) \\
&= 2\begin{bmatrix} 1 \\ e^{\frac{j\pi}{2}} \\ -1 \\ -e^{\frac{j\pi}{2}} \\ 1 \\ e^{\frac{j\pi}{2}} \\ -1 \\ -e^{\frac{j\pi}{2}} \end{bmatrix} + 3 - \begin{bmatrix} 1 \\ e^{\frac{j\pi}{4}} \\ e^{\frac{j\pi}{2}} \\ e^{\frac{j3\pi}{4}} \\ -1 \\ -e^{\frac{j\pi}{4}} \\ -e^{\frac{j\pi}{2}} \\ -e^{\frac{j3\pi}{4}} \end{bmatrix} \\
&= \begin{bmatrix} 5 \\ 2j+3 \\ 1 \\ -2j+3 \\ 5 \\ 2j+3 \\ 1 \\ -2j+3 \end{bmatrix} - \begin{bmatrix} 1 \\ e^{\frac{j\pi}{4}} \\ j \\ e^{\frac{j3\pi}{4}} \\ -1 \\ -e^{\frac{j\pi}{4}} \\ -j \\ -e^{\frac{j3\pi}{4}} \end{bmatrix} \\
&= \begin{bmatrix} 4 \\ \left(3 - \frac{1}{\sqrt{2}}\right) + \left(2 - \frac{1}{\sqrt{2}}\right) j \\ 1 - j \\ \left(3 + \frac{1}{\sqrt{2}}\right) - \left(2 + \frac{1}{\sqrt{2}}\right) j \\ 6 \\ \left(3 + \frac{1}{\sqrt{2}}\right) + \left(2 + \frac{1}{\sqrt{2}}\right) j \\ 1 + j \\ \left(3 - \frac{1}{\sqrt{2}}\right) - \left(2 - \frac{1}{\sqrt{2}}\right) j \end{bmatrix}
\end{aligned}$$

We then calculate $C(X)$ by multiplying the corresponding elements in $A(X)$ and $B(X)$. I got bored and wrote a program to do this. The result is

$$\begin{aligned}
C(X) &= A(X). * B(X) \\
&\approx \begin{bmatrix} 32 \\ 9.7867966 + 7.8700577j \\ 12 - 10j \\ 52.213203 + 45.870058j \\ -84 \\ 52.213203 - 45.870058j \\ 12 + 10j \\ 9.7867966 - 7.8700577j \end{bmatrix}
\end{aligned}$$

We then calculate $D(X) = \sum_{i=0}^{2n-1} C(X)_i X^i$

$$\frac{D(X)}{2n} = \begin{bmatrix} 0 \\ 0 \\ 10 \\ -19 \\ 34 \\ -19 \\ 14 \\ 12 \end{bmatrix}$$

You can see the coefficients in order. Compare with the results obtained in the beginning of the chapter. While this is slow and tedious for people, it is very fast for a machine, which excels at such mundane calculations.

Chapter 7

Laplace Transform

We will concentrate on the Laplace transform as the Fourier transform is a special case where $s = jw$ (note: I am an engineer so I use $j = \sqrt{-1}$ instead of i and i is current). The two sided Laplace transform is defined by the integral transform

$$F(s) = \int_{-\infty}^{\infty} f(t)e^{-st}dt.$$

The one sided Laplace transform is defined by the integral transform

$$F(s) = \int_{0}^{\infty} f(t)e^{-st}dt.$$

Both have their reasons, but the one-sided transform is usually the one which is used. It has advantages of handling initial conditions, and some more subtle areas too detailed to go into right now. One key area to consider is the convergence of the integral. Does the integral even exist? For instance what if $f(t) = 1$ and $s = 0$? In this case we have the integral of 1 over an infinite length, and the integral does not converge (exist). In general s is restricted to the values in the complex plane, which will allow convergence. The question may arise as to the utility of the Laplace transform. Its main utility is in the ease of use, and the way it simplifies problems. Lets find some Laplace transform properties and then we will find some transform pairs and finally we will see how to use the Laplace transform on practical problems.

7.1 Properties

$$\mathcal{L}\left(f(t)\right) = F(s) = \int_{0}^{\infty} f(t)e^{-st}dt$$

7.1.1 Derivatives

This is a simple case of integration by parts.

$$\begin{aligned}\mathcal{L}\left(f'(t)\right) &= \int_0^\infty f'(t)e^{-st}dt \\ &= (f(t)e^{-st})|_0^\infty + s\int_0^\infty f(t)e^{-st}dt \\ &= sF(s) - f(0)\end{aligned}$$

While the proof is simple, the result is profound. Essentially this says that differentiation can be transformed into algebra. This and the convolution property are the reason Laplace (and thus Fourier) transforms are so powerful.

7.1.2 Integration

This is done by changing the order of integration.

$$\begin{aligned}\mathcal{L}\left(\int_0^t f(\tau)d\tau\right) &= \int_0^\infty \int_0^t f(\tau)d\tau e^{-st}dt \\ &= \int_0^\infty \int_\tau^\infty f(\tau)e^{-st}dtd\tau \\ &= \int_0^\infty f(\tau)\int_\tau^\infty e^{-st}dtd\tau \\ &= \int_0^\infty f(\tau)\frac{1}{-s}e^{-st}\|_\tau^\infty d\tau \\ &= \int_0^\infty f(\tau)\frac{1}{s}e^{-s\tau}d\tau \\ &= \frac{1}{s}\int_0^\infty f(\tau)e^{-s\tau}d\tau \\ &= \frac{1}{s}F(s)\end{aligned}$$

This shows the other half of what we showed with derivatives - namely that calculus has become algebra.

7.1.3 Convolution

$$\begin{aligned}\mathcal{L}\left(f(t) * g(t)\right) &= \mathcal{L}\left(\int_0^\infty f(\tau)g(t-\tau)d\tau\right)\\ &= \int_0^\infty \int_0^\infty f(\tau)g(t-\tau)d\tau e^{-st}dt\\ &= \int_0^\infty f(\tau)\int_0^\infty g(t-\tau)e^{-st}dtd\tau\end{aligned}$$

Now, let $\zeta = t - \tau$, thus $t = \zeta + \tau$ and $d\zeta = dt$.

$$\begin{aligned}\mathcal{L}\left(f(t) * g(t)\right) &= \int_0^\infty f(\tau)\int_0^\infty g(t-\tau)e^{-st}dtd\tau\\ &= \int_0^\infty f(\tau)\int_0^\infty g(\zeta)e^{-s(\zeta+\tau)}d\zeta d\tau\\ &= \int_0^\infty f(\tau)\int_0^\infty g(\zeta)e^{-s\zeta}e^{-s\tau}d\zeta d\tau\\ &= \int_0^\infty f(\tau)\int_0^\infty g(\zeta)e^{-s\zeta}d\zeta e^{-s\tau}d\tau\\ &= \int_0^\infty f(\tau)e^{-s\tau}d\tau\int_0^\infty g(\zeta)e^{-s\zeta}d\zeta\\ &= \mathcal{L}\left(f(t)\right)\mathcal{L}\left(g(t)\right)\\ &= F(s)G(s)\end{aligned}$$

Convolution becomes multiplication.

7.2 Transform Pairs

$f(t)$	$\leftrightarrow$	$F(s)$
$\delta(t)$	$\leftrightarrow$	1
$u(t)$	$\leftrightarrow$	$\frac{1}{s}$
$e^{-at}u(t)$	$\leftrightarrow$	$\frac{1}{s+a}$
$\sin(\omega t)u(t)$	$\leftrightarrow$	$\frac{\omega}{s^2+\omega^2}$
$\cos(\omega t)u(t)$	$\leftrightarrow$	$\frac{s}{s^2+\omega^2}$
$e^{-at}f(f)$	$\leftrightarrow$	$F(s+a)$
t^n	$\leftrightarrow$	$\frac{n!}{s^{n+1}}$
$f^{(k)}(t)$	$\leftrightarrow$	$s^kF(s) - s^{k-1}f(o^+) - s^{k-2}f'(0^+) - \ldots - f^{(k-1)}(0^+)$
$\int_{-\infty}^t f(\tau)d\tau$	$\leftrightarrow$	$\frac{F(s)}{s} + \frac{\int_{-\infty}^0 f(\tau)d\tau}{s}$

7.3 Inverse Laplace Transform

The inverse Laplace transform is is given by

$$f(t) = \frac{1}{2\pi j}\int_{\sigma-j\infty}^{\sigma+j\infty} F(s)e^{st}ds.$$

We rarely compute the inverse laplace transform directly, however, as it is easier to use the Heavyside partial fraction expansion to calculate the residue, and then use the table of transform pairs. This is best illustrated with an example.

Example 3 *consider a RLC circuit given by:*

$$L\frac{d^2}{dt^2}q(t) + R\frac{d}{dt}q(t) + \frac{1}{c}q(t) = v(t) \tag{7.1}$$

What is the charge, $q(t)$, assuming the system is initially at rest, the voltage is a step function (light switch is turned on), and $L = 1$, $R = 250$, and $C = 10^{-4}$?

Solution:

First, take the Laplace Transform.

$$Ls^2Q(s) - Lsq(0^+) - Lq'(0^+) + RsQ(s) - Rq(0^+) + \frac{1}{c}Q(s) = V(s) \tag{7.2}$$

$$s^2Q(s) + 250sQ(s) + 10000Q(s) = \frac{1}{s} \tag{7.3}$$

$$(s^2 + 250s + 10000)Q(s) = \frac{1}{s} \tag{7.4}$$

$$Q(s) = \frac{1}{s(s^2 + 250s + 10000)} \tag{7.5}$$

$$Q(s) = \frac{1}{s(s + 200)(s + 50)} \tag{7.6}$$

In partial fraction expansion we look for a solution of the form:

$$Q(s) = \frac{r_1}{s} + \frac{r_2}{s + 200} + \frac{r_3}{s + 50} \tag{7.7}$$

thus

$$1 = r_1(s + 200)(s + 50) + r_2s(s + 50) + r_3s(s + 200). \tag{7.8}$$

This must be true for all s, so it must hold true for $s \in \{0, -50, -200\}$, so

$$1 = r_1(200)(50) \tag{7.9}$$

$$r_1 = 10^{-4} \tag{7.10}$$

$$1 = r_2(-200)(-200 + 50) \tag{7.11}$$

$$r_2 = \frac{1}{3} \times 10^{-4} \tag{7.12}$$

$$1 = r_3(-50)(-50 + 200) \tag{7.13}$$

$$r_3 = -\frac{4}{3} \times 10^{-4}. \tag{7.14}$$

This means that

$$Q(s) = \frac{10^{-4}}{s} + \frac{\frac{1}{3}\times 10^{-4}}{s+200} - \frac{\frac{4}{3}\times 10^{-4}}{s+50}, \tag{7.15}$$

and so,

$$q(t) = 10^{-4}u(t) + \frac{1}{3}\times 10^{-4}e^{-200t} - \frac{4}{3}\times 10^{-4}e^{-50t}. \tag{7.16}$$

Example 4 *consider a spring-mass-damper system given by:*

$$m\frac{d^2}{dt^2}x(t) + c\frac{d}{dt}x(t) + kx(t) = 0 \tag{7.17}$$

Find the displacement, $x(t)$, assuming the system is initially at displaced to $x(0) = 1$ and $x'(0) = 0$ then released. Let $m = 1$, $c = 2$, and $k = 1$.

Solution:

First, take the Laplace Transform.

$$ms^2X(s) - msx(0) - mx'(0) + csX(s) - cx(0) + kX(s) = 0 \tag{7.18}$$
$$ms^2X(s) - ms + csX(s) - c + kX(s) = 0 \tag{7.19}$$
$$(ms^2 + cs + k)X(s) = ms + c \tag{7.20}$$
$$X(s) = \frac{ms+c}{ms^2+cs+k} \tag{7.21}$$
$$X(s) = \frac{s+2}{s^2+2s+1} \tag{7.22}$$

In partial fraction expansion we look for a solution of the form:

$$Q(s) = \frac{r_1}{s} + \frac{r_2}{s+200} + \frac{r_3}{s+50} \tag{7.23}$$

thus

$$1 = r_1(s+200)(s+50) + r_2 s(s+50) + r_3 s(s+200). \tag{7.24}$$

This must be true for all s, so it must hold true for $s \in \{0, -50, -200\}$, so

$$1 = r_1(200)(50) \tag{7.25}$$
$$r_1 = 10^{-4} \tag{7.26}$$
$$1 = r_2(-200)(-200+50) \tag{7.27}$$
$$r_2 = \frac{1}{3}\times 10^{-4} \tag{7.28}$$
$$1 = r_3(-50)(-50+200) \tag{7.29}$$
$$r_3 = -\frac{4}{3}\times 10^{-4}. \tag{7.30}$$

This means that

$$Q(s) = \frac{10^{-4}}{s} + \frac{\frac{1}{3}\times 10^{-4}}{s+200} - \frac{\frac{4}{3}\times 10^{-4}}{s+50}, \tag{7.31}$$

and so,

$$q(t) = 10^{-4}u(t) + \frac{1}{3}\times 10^{-4}e^{-200t} - \frac{4}{3}\times 10^{-4}e^{-50t}. \tag{7.32}$$

7.4 Root Locus

Definitions:

Plant	$G(s)$
Gain	K
Measurements	$H(s)$
Transfer function	$T(s) = \frac{KG(s)}{1+KG(s)H(s)}$
Characteristic equation	$1 + KG(s)H(s) = 0$
Gain criteria	$\lvert KG(s)H(s)\rvert = 1$
Phase criteria	$\angle G(s)H(s) = (2n+1)\pi, \forall n \in \mathbb{Z}$
Number of Poles	P
Number of Zeros	Z

Properties/Basics:

- Characteristic equation must be satisfied for all points on the locus.
- The locus is symmetric about the real axis.
- The "open loop poles" are the poles of $G(s)H(s)$.
- The "open loop zeros" are the zeros of $G(s)H(s)$.
- Each "open loop pole" generates a branch of the locus.
- Each branch ends at a zero or at infinity along an asymptote.
- The derivative of the characteristic equation is zero where branches meet.

Drawing:

1. plot the open loop poles and zeros
2. On the real axis the locus exists to the left of an odd number of real poles or zeros.
3. The asymptotes are defined by their angle and center as follows:

$$center = \frac{\sum_{i=0}^{P-1} p_i - \sum_{i=0}^{Z-1} z_i}{P-Z} \tag{7.33}$$

$$angle = \frac{2n+1}{P-Z}\pi \tag{7.34}$$

4. The break-away and break-in points can be calculated by the derivative condition. If $N(s)$ is the numerator of $G(s)H(s)$ and $D(s)$ is the denominator, then:

$$N(s)\dot{D}(s) = \dot{N}(s)D(s) \tag{7.35}$$

5. The angle of departure (arrival) at a complex pole (zero) is given by:

$$x \tag{7.36}$$

Chapter 8

Z Transform

8.1 One Sided Transform

$$F^+(z) = \lim_{k\to\infty} \sum_{i=0}^{k} f(i) z^{-i} \tag{8.1}$$

$f(n)$	$\leftrightarrow$	$F(z)$	ROC
$\delta(n)$	$\leftrightarrow$	1	
$u(n)$	$\leftrightarrow$	$\frac{1}{1-z^{-1}}$	$\|z\| > 1$
	$\leftrightarrow$	$\frac{z}{z-1}$	$\|z\| > 1$
$a^n u(n)$	$\leftrightarrow$	$\frac{1}{1-az^{-1}}$	$\|z\| > \|a\|$
	$\leftrightarrow$	$\frac{z}{z-a}$	$\|z\| > \|a\|$
$\alpha f(n)$	$\leftrightarrow$	$\alpha F(z)$	ROC of $F(z)$
$f(n) + g(n)$	$\leftrightarrow$	$F(z) + G(z)$	ROC of $F(z) \bigcap$ ROC of $G(z)$
$f(n) * g(n)$	$\leftrightarrow$	$F(z)G(z)$	ROC of $F(z) \bigcap$ ROC of $G(z)$
$f(n-k)$	$\leftrightarrow$	$z^{-k}F(z)$	ROC of $F(z)$

8.2 Partial Fractions

8.2.1 Example

Find the output, $y(n)$, of a system, $h(n)$, driven by an input, $x(n)$, given:

$$y(n) = h(n) * x(n) \tag{8.2}$$

$$x(n) = u(n) \tag{8.3}$$

$$h(n) = \left(\frac{1}{3}\right)^n u(n) \tag{8.4}$$

Solution:

First, we take the Z transform

$$\begin{aligned} Y(z) &= H(z)X(z) && (8.5)\\ X(z) &= \frac{1}{1-z^{-1}} && (8.6)\\ H(z) &= \frac{1}{1-\frac{1}{3z}} && (8.7) \end{aligned}$$

thus

$$\begin{aligned} Y(z) &= \frac{1}{1-\frac{1}{3z}}\frac{1}{1-z^{-1}} && (8.8)\\ &= \frac{z}{z-\frac{1}{3}}\frac{z}{z-1} && (8.9)\\ & && (8.10)\\ &= \frac{z^2}{\left(z-\frac{1}{3}\right)(z-1)} && (8.11)\\ &= \frac{z^2}{z^2-\frac{4}{3}z+\frac{1}{3}} && (8.12)\\ &= \frac{z^2-\frac{4}{3}z+\frac{1}{3}+\frac{4}{3}z-\frac{1}{3}}{z^2-\frac{4}{3}z+\frac{1}{3}} && (8.13)\\ &= 1+\frac{\frac{4}{3}z-\frac{1}{3}}{z^2-\frac{4}{3}z+\frac{1}{3}} && (8.14)\\ &= 1+\frac{A}{z-\frac{1}{3}}+\frac{B}{z-1} && (8.15) \end{aligned}$$

Since the last two lines are equal and canceling the ones, we can write

$$\frac{\frac{4}{3}z-\frac{1}{3}}{z^2-\frac{4}{3}z+\frac{1}{3}} = \frac{A}{z-\frac{1}{3}}+\frac{B}{z-1} \quad (8.16)$$

thus

$$\frac{4}{3}z - \frac{1}{3} = A(z-1) + B\left(z - \frac{1}{3}\right) \tag{8.17}$$

$$B = 1.5 \tag{8.18}$$

$$A = -\frac{1}{6} \tag{8.19}$$

$$Y(z) = 1 + \frac{-\frac{1}{6}}{z - \frac{1}{3}} + \frac{1.5}{z-1} \tag{8.20}$$

$$Y(z) = 1 + \frac{-\frac{1}{6}zz^{-1}}{z - \frac{1}{3}} + \frac{1.5zz^{-1}}{z-1} \tag{8.21}$$

$$Y(z) = 1 - \frac{1}{6}z^{-1}\frac{z}{z - \frac{1}{3}} + 1.5z^{-1}\frac{z}{z-1} \tag{8.22}$$

$$y(n) = \delta(n) + 1.5u(n-1) - \frac{\left(\frac{1}{3}\right)^{n-1}u(n-1)}{6} \tag{8.23}$$

Now let's compare them. Consider theSciLab code in Code 8.1. The resulting solutions are in Fig 8.1 and show that the two methods generate the same solutions.

Listing 8.1: Code to test Z-transform solution.

```
// Keith Evan Schubert
// October 10, 2007
//
// This is a comparison of the z transform solution to directly
// calculating the convolution for
//      u(n) * (1/3)^nu(n)
//
// z transform was calculated to be
//     \delta(n)+1.5u(n-1)-((1/3)^{n-1}u(n-1))/6

// first I pick the lenght of the function I want to simulate
n=50;

// now declare my input
x=ones(n,1);

// declare my system
h=ones(n,1);
for i=2:n
  h(i)=h(i-1)/3;
end

// calculate the direct solution
y1=convol(h,x);
```

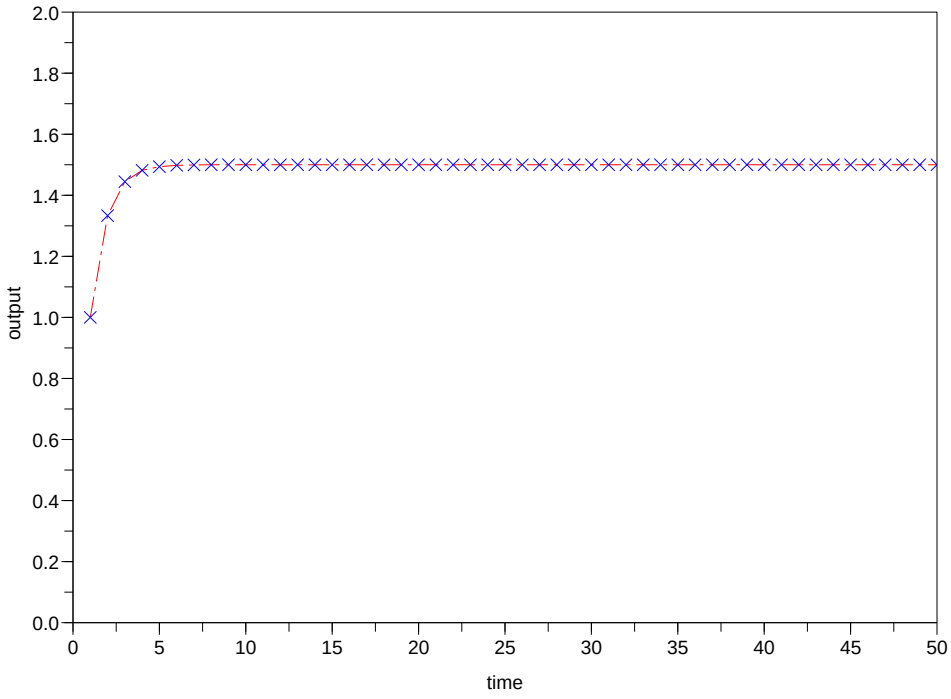
```
// Set up the size of the z-transform to be the same as y1
y2=zeros(2*n-1,1);

// put in the delta function
y2(1)=1;

// now add the step function and exponential function
temp1=-1/6;
for i=2:n
  y2(i)=1.5+temp1;
  temp1=temp1/3;
end

// plot them to compare, note only the first n are
// good due to truncation errors.
t=1:n;
plot(t,y1(1:n),"r:",t,y2(1:n),"bx")
xtitle("","time","output")
a=gca();
a.data_bounds=[0,0;n,2];
```

Figure 8.1: Comparison of Z-transform and convolution solutions to $y(n) = h(n) * x(n)$.



Chapter 9

Fourier Transform

9.1 Fourier Transform

Forward

$$X(f) = \int_{-\infty}^{\infty} x(t) e^{-2\pi\sqrt{-1}tf} dt \tag{9.1}$$

Inverse

$$x(t) = \int_{-\infty}^{\infty} X(f) e^{2\pi\sqrt{-1}tf} df \tag{9.2}$$

9.1.1 Standard Form

This comes from changing variables from f to ω

Forward

$$X(\omega) = \int_{-\infty}^{\infty} x(t) e^{-\sqrt{-1}t\omega} dt \tag{9.3}$$

Inverse

$$x(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} X(\omega) e^{\sqrt{-1}t\omega} d\omega \tag{9.4}$$

Conveniently this is the bilateral Laplace transform with $s = \sqrt{-1}\omega$.

9.1.2 Unitary Form

Forward

$$X(\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} x(t) e^{-\sqrt{-1}t\omega} dt \tag{9.5}$$

Inverse

$$x(t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} X(\omega) e^{\sqrt{-1}t\omega} d\omega \tag{9.6}$$

9.2 Discrete Fourier Transform

Discrete Fourier Transform

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk} \tag{9.7}$$

$$W_N = e^{\frac{2\pi\sqrt{-1}}{N}} \tag{9.8}$$

9.3 Fast Fourier Transform

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk} \tag{9.9}$$

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk} \tag{9.10}$$

Part III

Estimation

Chapter 10

Least Squares

Suppose we want to solve the following system

$$Ax \approx b \tag{10.1}$$

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].

$$\hat{x} = \text{argmin}_x \|Ax - b\| \tag{10.2}$$

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)

$$\begin{aligned}
\nabla_x \|Ax - b\|^2 &= \nabla_x \left((Ax - b)^T (Ax - b)\right) && (10.3)\\
&= \nabla_x \left(x^T A^T Ax - 2b^T Ax + b^T b)\right) && (10.4)\\
&= 2A^T Ax - 2A^T b && (10.5)\\
&= 2A^T (Ax - b) && (10.6)
\end{aligned}$$

At the minimum the derivative must be zero thus

$$\begin{aligned}
0 &= 2A^T (Ax - b) && (10.7)\\
0 &= A^T Ax - A^T b && (10.8)\\
A^T Ax &= A^T b && (10.9)\\
x &= (A^T A)^{-1} A^T b && (10.10)
\end{aligned}$$

[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 10.9 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.

10.1 Recursive Least Squares

$$A_{k+1}x_{k+1} = b_{k+1} \tag{10.11}$$

$$A_{k+1} = \begin{bmatrix} A_k \\ a_{k+1}^T \end{bmatrix}, \quad b_{k+1} = \begin{bmatrix} b_k \\ \beta_{k+1} \end{bmatrix} \tag{10.12}$$

$$\begin{bmatrix} A_k \\ a_{k+1}^T \end{bmatrix} x_{k+1} = \begin{bmatrix} b_k \\ \beta_{k+1} \end{bmatrix} \tag{10.13}$$

Thus

$$A_{k+1}^T A_{k+1} x_{k+1} = A_{k+1}^T b_{k+1} \tag{10.14}$$

$$\begin{bmatrix} A_k \\ a_{k+1}^T \end{bmatrix}^T \begin{bmatrix} A_k \\ a_{k+1}^T \end{bmatrix} x_{k+1} = \begin{bmatrix} A_k \\ a_{k+1}^T \end{bmatrix}^T \begin{bmatrix} b_k \\ \beta_{k+1} \end{bmatrix} \tag{10.15}$$

$$(A_k^T A_k + a_{k+1}a_{k+1}^T)x_{k+1} = A_k^T b_k + a_{k+1}\beta_{k+1} \tag{10.16}$$

and

$$x_{k+1} = (A_{k+1}^T A_{k+1})^{-1} A_{k+1}^T b_{k+1} \tag{10.17}$$

$$= (A_k^T A_k + a_{k+1}a_{k+1}^T)^{-1}(A_k^T b_k + a_{k+1}\beta_{k+1}) \tag{10.18}$$

Now if we define

$$P_{k+1} = (A_{k+1}^T A_{k+1})^{-1} \tag{10.19}$$

$$= (A_k^T A_k + a_{k+1}a_{k+1}^T)^{-1} \tag{10.20}$$

[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

$$\begin{aligned} P_{k+1}^{-1} &= A_{k+1}^T A_{k+1} && (10.21)\\ &= A_k^T A_k + a_{k+1} a_{k+1}^T && (10.22)\\ &= P_k^{-1} + a_{k+1} a_{k+1}^T && (10.23)\\ x_{k+1} &= P_{k+1} A_{k+1}^T b_{k+1} && (10.24)\\ P_{k+1}^{-1} x_{k+1} &= A_{k+1}^T b_{k+1} && (10.25) \end{aligned}$$

Next take Eq 10.24

$$\begin{aligned} x_{k+1} &= P_{k+1} A_{k+1}^T b_{k+1} && (10.26)\\ &= P_{k+1}(A_k^T b_k + a_{k+1}\beta_{k+1}) && (10.27)\\ &= P_{k+1}(P_k^{-1} x_k + a_{k+1}\beta_{k+1}) && (10.28)\\ &= P_{k+1}((P_{k+1}^{-1} - a_{k+1}a_{k+1}^T)x_k + a_{k+1}\beta_{k+1}) && (10.29)\\ &= P_{k+1}(P_{k+1}^{-1} x_k - a_{k+1}a_{k+1}^T x_k + a_{k+1}\beta_{k+1}) && (10.30)\\ &= x_k + P_{k+1}(a_{k+1}\beta_{k+1} - a_{k+1}a_{k+1}^T x_k) && (10.31)\\ &= x_k + P_{k+1}a_{k+1}(\beta_{k+1} - a_{k+1}^T x_k) && (10.32)\\ &= x_k + K_{k+1}(\beta_{k+1} - a_{k+1}^T x_k) && (10.33) \end{aligned}$$

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

$$\begin{aligned} x_{k+1} &= x_k + K_{k+1}(\beta_{k+1} - a_{k+1}^T x_k) && (10.34)\\ K_{k+1} &= P_{k+1} a_{k+1} && (10.35)\\ P_{k+1}^{-1} &= P_k^{-1} + a_{k+1} a_{k+1}^T && (10.36) \end{aligned}$$

10.1.1 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 10.1. The test code has a few things worth mentioning, see Code 10.2. 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 10.1 and Fig 10.2. Notice that the solution converges to the real value.

Listing 10.1: 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 10.2: 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]")
```

10.2 Covariance Form

Lemma 1 (Matrix Inversion) *If*

$$A = B + C^T DC \tag{10.37}$$

Then

$$A^{-1} = B^{-1} - B^{-1}C^T \left(CB^{-1}C^T + D^{-1}\right)^{-1} CB^{-1} \tag{10.38}$$

Figure 10.1: Comparisons of "True" parameter values versus estimates for several randomly generated problems

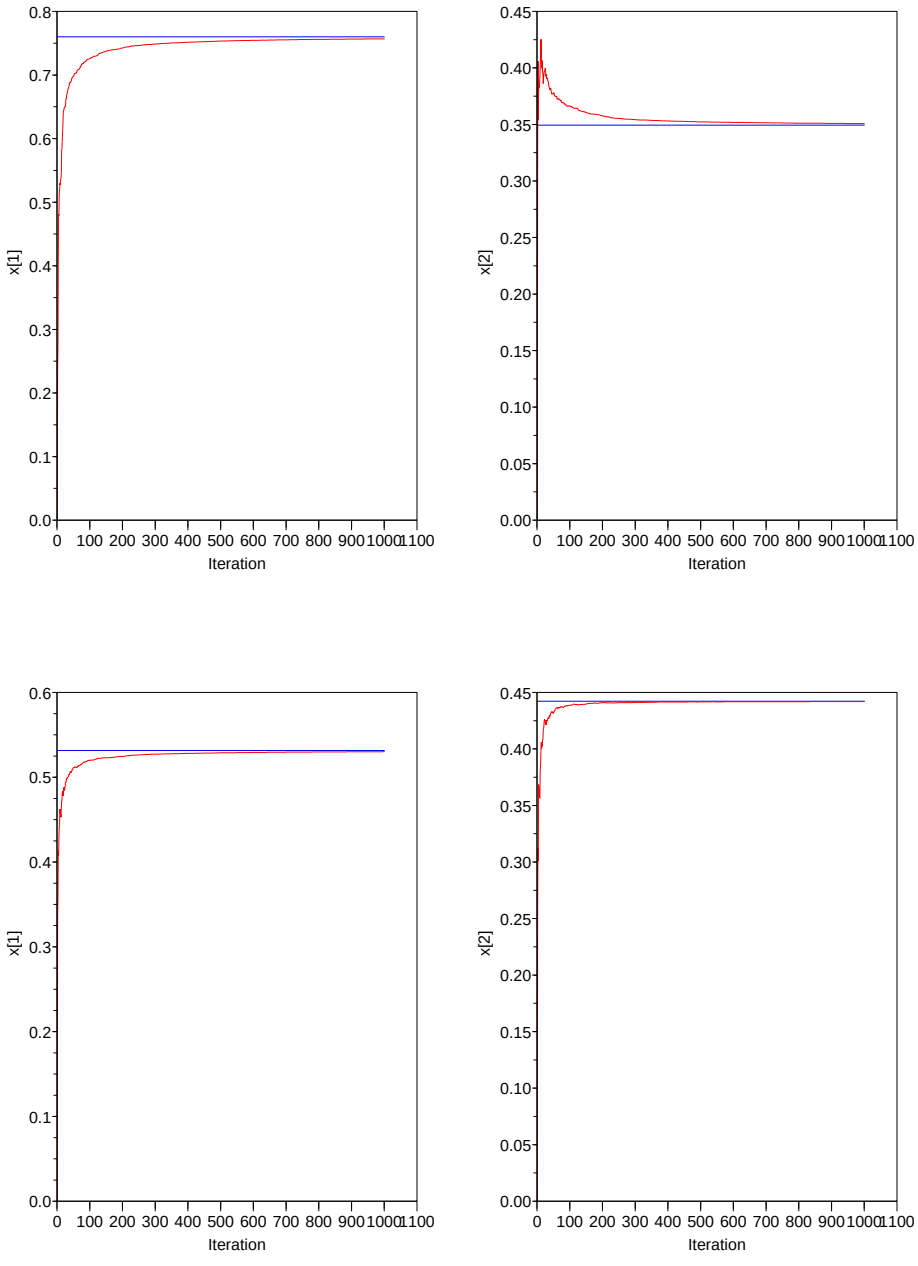
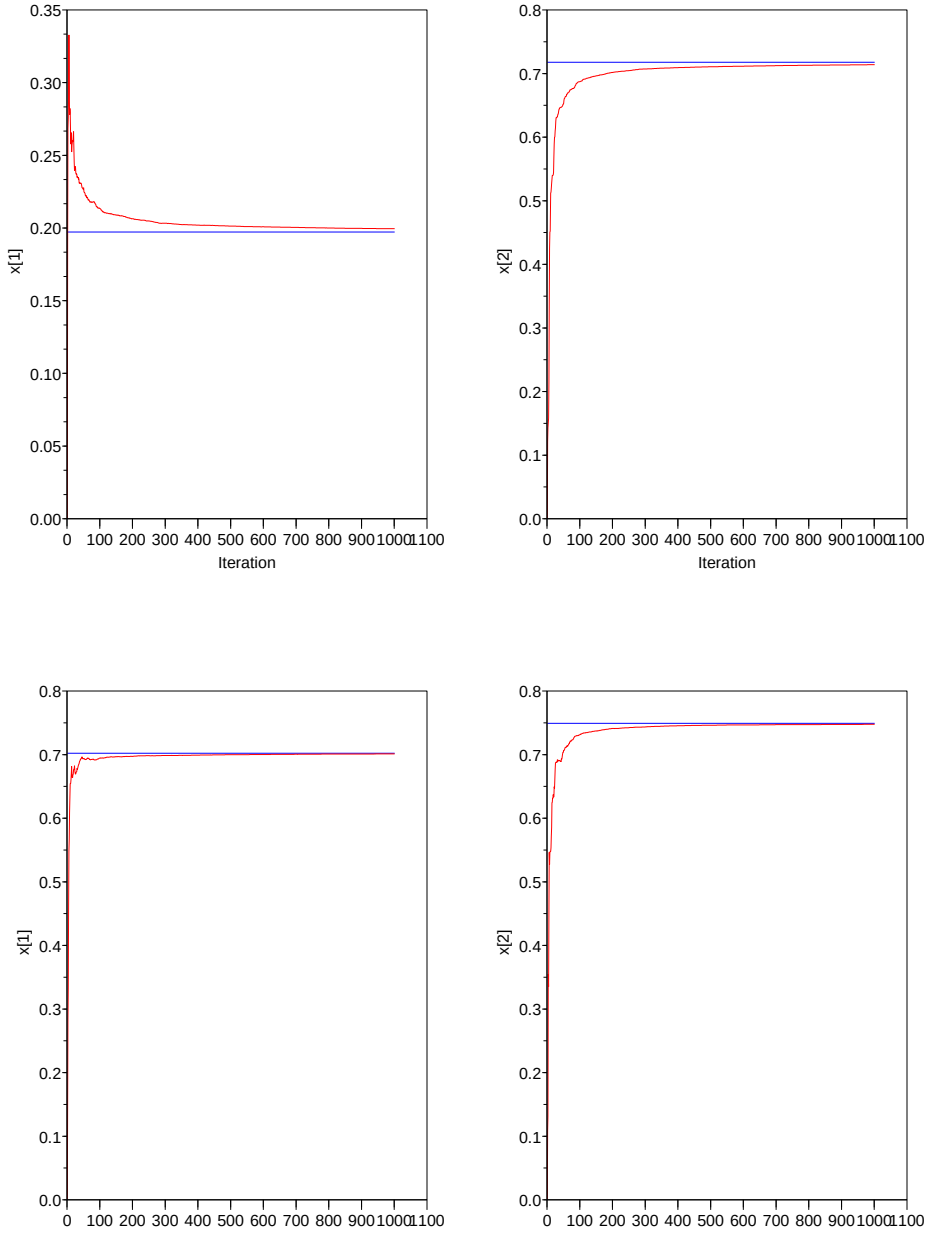


Figure 10.2: More comparisons of "True" parameter values versus estimates for several randomly generated problems



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

$$P_{k+1} = P_k - P_k a_{k+1} \left(a_{k+1}^T P_k a_{k+1} + 1\right)^{-1} a_{k+1}^T P_k \tag{10.39}$$

Now note that

$$\begin{aligned}
K_{k+1} &= P_{k+1}a_{k+1} && (10.40)\\
&= P_k a_{k+1} - P_k a_{k+1} \left(a_{k+1}^T P_k a_{k+1} + 1\right)^{-1} a_{k+1}^T P_k a_{k+1} && (10.41)\\
&= P_k a_{k+1} - P_k a_{k+1} \left(a_{k+1}^T P_k a_{k+1} + 1\right)^{-1} (a_{k+1}^T P_k a_{k+1} + 1 - 1) && (10.42)\\
&= P_k a_{k+1} - P_k a_{k+1} + P_k a_{k+1} \left(a_{k+1}^T P_k a_{k+1} + 1\right)^{-1} && (10.43)\\
&= P_k a_{k+1} \left(a_{k+1}^T P_k a_{k+1} + 1\right)^{-1} && (10.44)
\end{aligned}$$

Using this we have

$$\begin{aligned}
P_{k+1} &= P_k - K_{k+1} a_{k+1}^T P_k && (10.45)\\
&= (I - K_{k+1} a_{k+1}^T) P_k && (10.46)
\end{aligned}$$

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

$$\begin{aligned}
x_{k+1} &= x_k + K_{k+1}(\beta_{k+1} - a_{k+1}^T x_k) && (10.47)\\
K_{k+1} &= P_k a_{k+1} \left(a_{k+1}^T P_k a_{k+1} + 1\right)^{-1} && (10.48)\\
P_{k+1} &= (I - K_{k+1} a_{k+1}^T) P_k && (10.49)
\end{aligned}$$

10.2.1 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 10.3. The test code has a few things worth mentioning, see Code 10.4. 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 10.3 and Fig 10.4. Notice that the solution converges to the real value.

Listing 10.3: 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 10.4: 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]")
```

10.3 Estimation with Noise

$$y = Ax + v \tag{10.50}$$

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

$$\begin{aligned} E[y] &= E[Ax + v] && (10.51)\\ &= AE[x] + E[v] && (10.52)\\ &= A\bar{x} && (10.53) \end{aligned}$$

and their covariance is

$$\begin{aligned} E[(y - E[y])(y - E[y])^T] &= E[(A(x - \bar{x}) + v)(A(x - \bar{x}) + v)^T] && (10.54)\\ &= E[A(x - \bar{x})(x - \bar{x})^T A^T + vv^T] && (10.55)\\ &= AP_xA^T + P_v. && (10.56) \end{aligned}$$

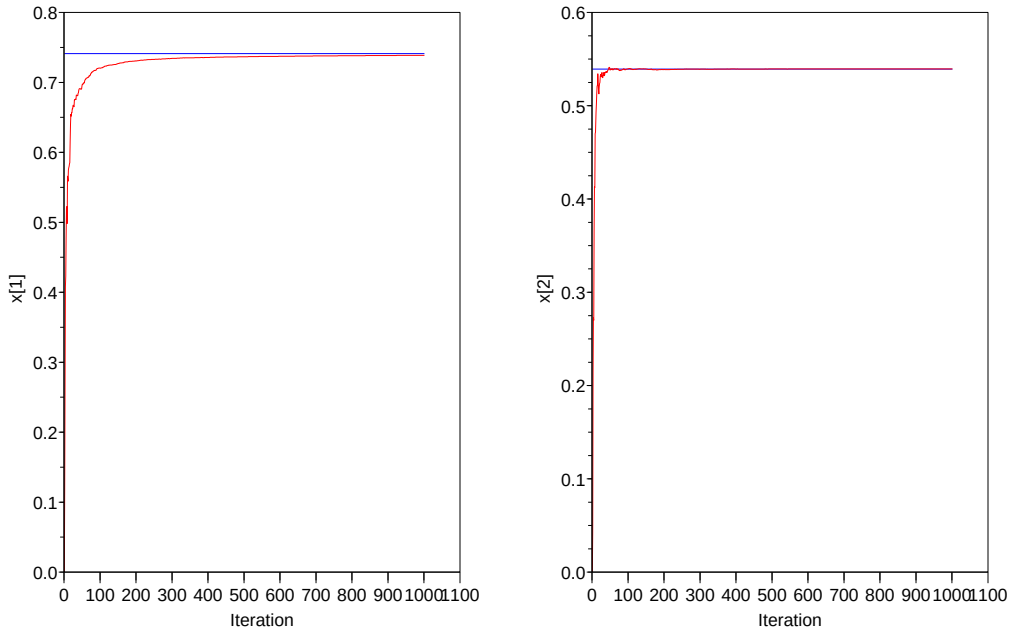
Now, note the covariance of x and y is

$$\begin{aligned} E[(x - E[x])(y - E[y])^T] &= E[(x - \bar{x})(A(x - \bar{x}) + v)^T] && (10.57)\\ &= E[(x - \bar{x})(x - \bar{x})^T A^T] && (10.58)\\ &= P_xA^T. && (10.59) \end{aligned}$$

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:

$$\begin{aligned} \hat{x}_{k+1} &= \hat{x}_k + K_k\nu_{k+1|k} && (10.60)\\ \nu_{k+1|k} &= y_{k+1} - A\hat{x}_k && (10.61) \end{aligned}$$

Figure 10.3: Comparisons of "True" parameter values versus estimates for several randomly generated problems



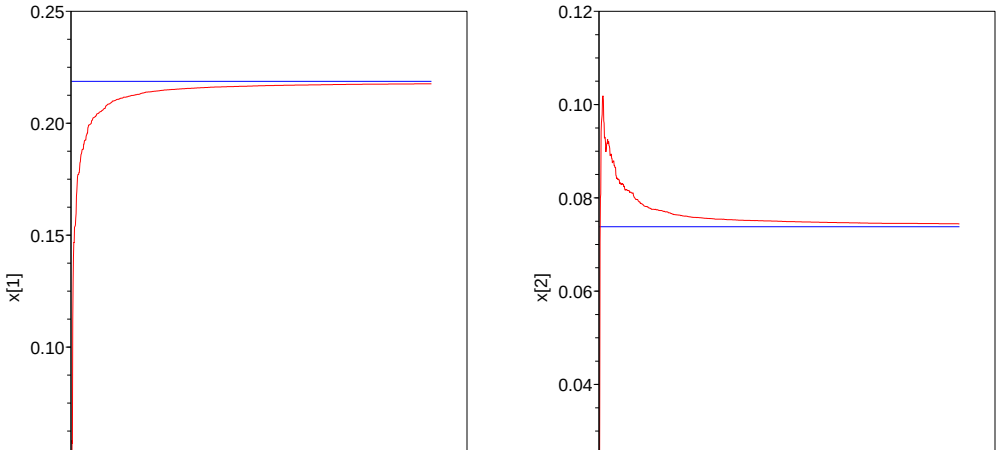
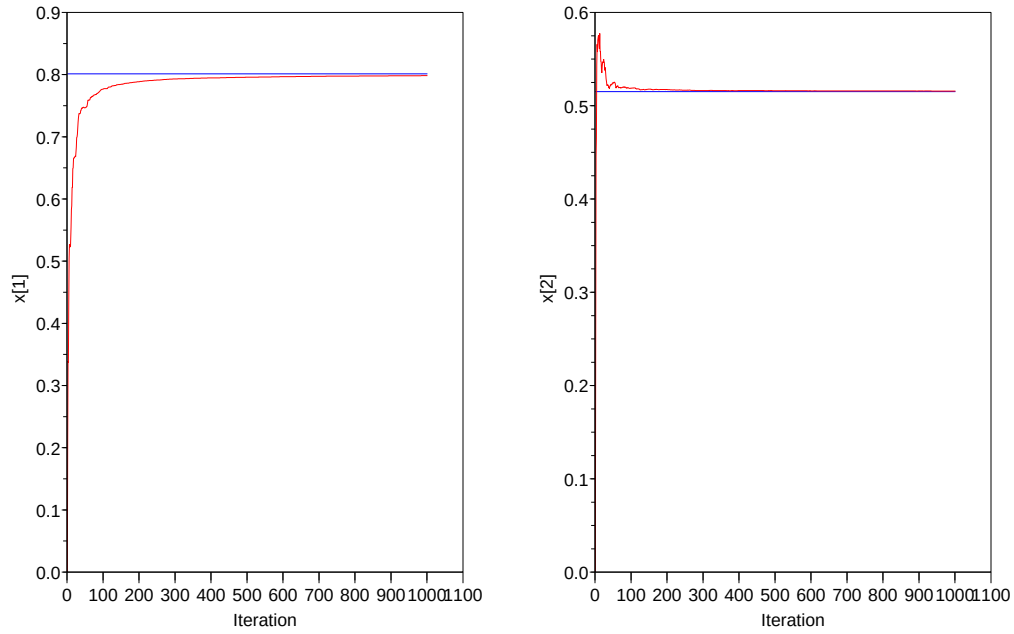
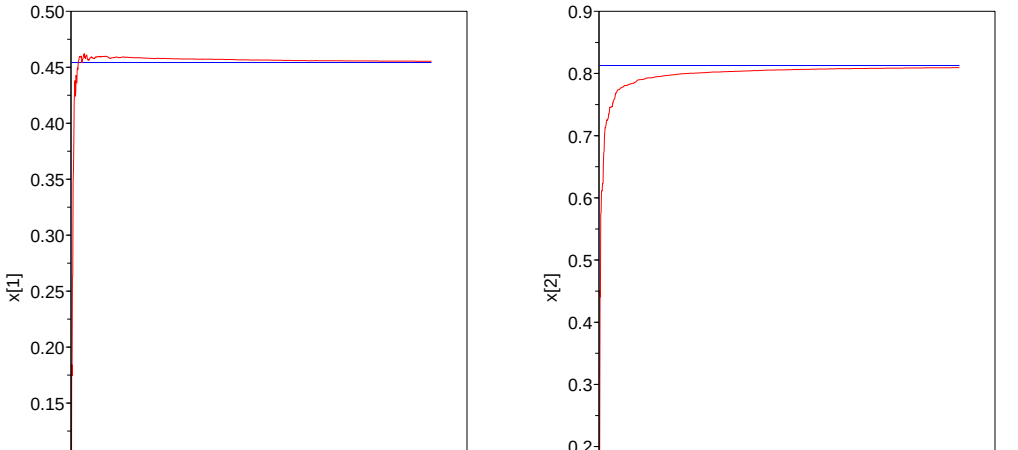


Figure 10.4: More comparisons of "True" parameter values versus estimates for several randomly generated problems





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

$$\begin{aligned} P_e k &= E[e_k e_k^T] && (10.62)\\ &= E[(x-\hat{x}_k)(x-\hat{x}_k)^T] && (10.63)\\ &= E[(x-\mu+\mu-\hat{x}_k)(x-\mu+\mu-\hat{x}_k)^T] && (10.64)\\ &= E[(x-\mu)(x-\mu)^T] + E[(x-\mu)(\mu-\hat{x}_k)^T] + E[(\mu-\hat{x}_k)(x-\mu)^T] + E[(\mu-\hat{x}_k)(\mu-\hat{x}_k)^T] && (10.65) \end{aligned}$$

Now

$$\begin{aligned} P_e k+1 &= E[e_{k+1} e_{k+1}^T] && (10.66)\\ &= E[(x-\hat{x}_{k+1})(x-\hat{x}_{k+1})^T] && (10.67)\\ &= E[(x-\hat{x}_k - K_k\nu_{k+1|k})(x-\hat{x}_k - K_k\nu_{k+1|k})^T] && (10.68)\\ &= E[(e_k - K_k\nu_{k+1|k})(e_k - K_k\nu_{k+1|k})^T] && (10.69) \end{aligned}$$

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

$$\begin{aligned} P_e k+1 &= E[(e_k - K_k\nu_{k+1|k})(e_k - K_k\nu_{k+1|k})^T] && (10.70)\\ &= E[e_k e_k^T] - E[e_k\nu_{k+1|k}^T K_k^T] - E[K_k\nu_{k+1|k}e_k^T] + E[K_k\nu_{k+1|k}\nu_{k+1|k}^T K_k^T] && (10.71)\\ &= P_e k - E[e_k\nu_{k+1|k}^T]K_k^T - K_k E[\nu_{k+1|k}e_k^T] + K_k E[\nu_{k+1|k}\nu_{k+1|k}^T]K_k^T && (10.72) \end{aligned}$$

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

$$\begin{aligned} E[\nu_{k+1|k}e_k^T] &= E[(y_{k+1}-A\hat{x}_k)e_k^T] && (10.73)\\ &= E[(Ax+v_{k+1}-A\hat{x}_k)e_k^T] && (10.74)\\ &= E[(Ae_k+v_{k+1})e_k^T] && (10.75)\\ &= AP_e k && (10.76) \end{aligned}$$

and

$$\begin{aligned} E[\nu_{k+1|k}\nu_{k+1|k}^T] &= E[(y_{k+1}-A\hat{x}_k)(y_{k+1}-A\hat{x}_k)^T] && (10.77)\\ &= E[(Ax+v_{k+1}-A\hat{x}_k)(Ax+v_{k+1}-A\hat{x}_k)^T] && (10.78)\\ &= E[(Ae_k+v_{k+1})(Ae_k+v_{k+1})^T] && (10.79)\\ &= AP_e kA^T + P_v && (10.80) \end{aligned}$$

Thus,

$$P_e k+1 = P_e k - P_e kA^T K_k^T - K_k AP_e k + K_k(AP_e kA^T + P_v)K_k^T \quad (10.81)$$

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

$$\begin{aligned} 0 &= -2P_e kA^T + 2K_k(AP_e kA^T + P_v) && (10.82)\\ K_k(AP_e kA^T + P_v) &= P_e kA^T && (10.83)\\ K_k &= P_e kA^T(AP_e kA^T + P_v)^{-1} && (10.84) \end{aligned}$$

Thus

$$\begin{aligned} \hat{x}_{k+1} &= \hat{x}_k + K_k(y_{k+1} - A\hat{x}_k) && (10.85)\\ K_k &= P_e k A^T (A P_e k A^T + P_v)^{-1} && (10.86)\\ P_e k + 1 &= P_e k - P_e k A^T K_k^T - K_k A P_e k + K_k (A P_e k A^T + P_v) K_k^T && (10.87) \end{aligned}$$

$$\begin{aligned} E[y - E[\hat{y}]] &= E[A(x - \hat{x}) + v] && (10.88)\\ &= AE[(x - \hat{x})] + E[v] && (10.89)\\ &= A\bar{x} && (10.90) \end{aligned}$$

Thus

$$\begin{aligned} \hat{x} &= E[x|y] && (10.91)\\ &= E[(y - E[y])(y - E[y])^T]^{-1} E[(x - E[x])(y - E[y])^T] && (10.92)\\ &= P_x A^T && (10.93) \end{aligned}$$

Chapter 11

Kalman Filtering

11.1 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

$$p(x) \sim N(\mu, \sigma^2). \tag{11.1}$$

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

$$\begin{aligned} E[(y - E[y])^2] &= E[(ax + b - E[ax + b])^2] && (11.2)\\ &= E[(ax + b - a\mu - b)^2] && (11.3)\\ &= E[(a(x-\mu))^2] && (11.4)\\ &= a^2 E[(x-\mu)^2] && (11.5)\\ &= a^2\sigma^2 && (11.6) \end{aligned}$$

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

11.2 Discrete Kalman Filter

Assume we have a state space representation

$$\begin{aligned} x_{k+1} &= A_k x_k + B_k u_k + w_k && (11.7)\\ y_k &= C_k x_k + D_k u_k + v_k && (11.8) \end{aligned}$$

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

$$x_{k+1} = A_k x_k + w_k \tag{11.9}$$
$$y_k = C_k x_k + v_k \tag{11.10}$$

The random variables are given by

$$p(w) \sim N(0, R_w) \tag{11.11}$$
$$p(v) \sim N(0, R_v) \tag{11.12}$$

The Kalman filter is a predictor-corrector system.

- Predict $\hat{x}_{k|k-1}$, using Eq. 11.9. 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. 11.10. 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.

$$e_{k|k-1} = \hat{x}_{k|k-1} - x_k \tag{11.13}$$
$$e_{k|k} = \hat{x}_{k|k} - x_k \tag{11.14}$$

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

$$P_{k|k-1} = E[e_{k|k-1} e_{k|k-1}^T] \tag{11.15}$$
$$P_{k|k} = E[e_{k|k} e_{k|k}^T] \tag{11.16}$$

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

11.2.1 Prediction Step

Considering our state equation

$$x_{k+1} = A_k x_k + w_k. \tag{11.17}$$

[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.

$$\begin{aligned}
\hat{x}_{k+1|k} &= E[x_{k+1}|k] && (11.18)\\
&= E[A_k x_k + w_k|k] && (11.19)\\
&= A_k E[x_k|k] + E[w_k] && (11.20)\\
&= A_k \hat{x}_{k|k} + 0 && (11.21)\\
&= A_k \hat{x}_{k|k}. && (11.22)
\end{aligned}$$

The error in this is

$$\begin{aligned}
e_{k+1|k} &= x_{k+1} - \hat{x}_{k+1|k} && (11.23)\\
&= A_k x_k + w_k - A_k \hat{x}_{k|k} && (11.24)\\
&= A_k e_{k|k} + w_k. && (11.25)
\end{aligned}$$

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

$$\begin{aligned}
E[e_{k+1|k}] &= E[A_k e_{k|k} + w_k] && (11.26)\\
&= E[A_k e_{k|k}] + E[w_k] && (11.27)\\
&= A_k E[e_{k|k}] + 0 && (11.28)\\
&= A_k E[e_{k|k}]. && (11.29)
\end{aligned}$$

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

$$\begin{aligned}
P_{k+1|k} &= E[e_{k+1|k} e_{k+1|k}^T] && (11.30)\\
&= E[(A_k e_{k|k} + w_k)(A_k e_{k|k} + w_k)^T] && (11.31)\\
&= E[(A_k e_{k|k} + w_k)(e_{k|k}^T A_k^T + w_k^T)] && (11.32)\\
&= E[A_k e_{k|k} e_{k|k}^T A_k^T + w_k e_{k|k}^T A_k^T + A_k e_{k|k} w_k^T + w_k w_k^T] && (11.33)\\
&= E[A_k e_{k|k} e_{k|k}^T A_k^T] + E[w_k e_{k|k}^T A_k^T] + E[A_k e_{k|k} w_k^T] + E[w_k w_k^T] && (11.34)\\
&= A_k E[e_{k|k} e_{k|k}^T] A_k^T + E[w_k e_{k|k}^T] A_k^T + A_k E[e_{k|k} w_k^T] + E[w_k w_k^T] && (11.35)\\
&= A_k P_{k|k} A_k^T + (0) A_k^T + A_k (0) + R_w && (11.36)\\
&= A_k P_{k|k} A_k^T + R_w && (11.37)
\end{aligned}$$

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

$$\begin{aligned}
\hat{x}_{k+1|k} &= A_k \hat{x}_{k|k} && (11.38)\\
P_{k+1|k} &= A_k P_{k|k} A_k^T + R_w && (11.39)
\end{aligned}$$

11.2.2 Correction

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

$$y_k = C_k x_k + v_k \tag{11.40}$$

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.

$$\begin{aligned}
\hat{y}_{k|k-1} &= E[y_k|k-1] && (11.41)\\
&= E[C_k x_k + v_k|k-1] && (11.42)\\
&= E[C_k x_k|k-1] + E[v_k] && (11.43)\\
&= C_k E[x_k|k-1] + 0 && (11.44)\\
&= C_k \hat{x}_{k|k-1} && (11.45)
\end{aligned}$$

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

$$\begin{aligned}
\nu_{k|k-1} &= y_k - \hat{y}_{k|k-1} && (11.46)\\
&= y_k - C_k \hat{x}_{k|k-1} && (11.47)\\
&= C_k x_k + v_k - C_k \hat{x}_{k|k-1} && (11.48)\\
&= C_k e_{k|k-1} + v_k && (11.49)
\end{aligned}$$

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.

$$\begin{aligned}
\hat{x}_{k|k} &= \hat{x}_{k|k-1} + K_k \nu_{k|k-1} && (11.50)\\
&= \hat{x}_{k|k-1} + K_k (y_k - C_k \hat{x}_{k|k-1}) && (11.51)
\end{aligned}$$

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).

$$\begin{aligned}
e_{k|k} &= x_k - \hat{x}_{k|k} && (11.52)\\
&= x_k - \hat{x}_{k|k-1} - K_k \nu_{k|k-1} && (11.53)\\
&= e_{k|k-1} - K_k \nu_{k|k-1} && (11.54)
\end{aligned}$$

$$
\begin{aligned}
P_{k|k} &= E[e_{k|k}e_{k|k}^T] && (11.55)\\
&= E[(e_{k|k-1} - K_k\nu_{k|k-1})(e_{k|k-1} - K_k\nu_{k|k-1})^T] && (11.56)\\
&= E[e_{k|k-1}e_{k|k-1}^T] - E[e_{k|k-1}\nu_{k|k-1}^T K_k^T] - E[K_k\nu_{k|k-1}e_{k|k-1}^T]\\
&\quad + E[K_k\nu_{k|k-1}\nu_{k|k-1}^T K_k^T] && (11.57)\\
&= P_{k|k-1} - E[e_{k|k-1}(C_k e_{k|k-1} + v_k)^T K_k^T] - E[K_k(C_k e_{k|k-1} + v_k)e_{k|k-1}^T]\\
&\quad + E[K_k(C_k e_{k|k-1} + v_k)(C_k e_{k|k-1} + v_k)^T K_k^T] && (11.58)\\
&= P_{k|k-1} - E[e_{k|k-1}e_{k|k-1}^T C_k^T K_k^T + e_{k|k-1}v_k^T K_k^T]\\
&\quad - E[K_k C_k e_{k|k-1}e_{k|k-1}^T + K_k v_k e_{k|k-1}^T] + E[K_k C_k e_{k|k-1}e_{k|k-1}^T C_k^T K_k^T\\
&\quad + K_k C_k e_{k|k-1}v_k K_k^T + K_k v_k e_{k|k-1}^T C_k^T K_k^T + K_k v_k v_k K_k^T] && (11.59)\\
&= P_{k|k-1} - E[e_{k|k-1}e_{k|k-1}^T C_k^T K_k^T] + E[e_{k|k-1}v_k^T K_k^T] - E[K_k C_k e_{k|k-1}e_{k|k-1}^T]\\
&\quad + E[K_k v_k e_{k|k-1}^T] + E[K_k C_k e_{k|k-1}e_{k|k-1}^T C_k^T K_k^T]\\
&\quad + E[K_k C_k e_{k|k-1}v_k K_k^T] + E[K_k v_k e_{k|k-1}^T C_k^T K_k^T] + E[K_k v_k v_k K_k^T] && (11.60)\\
&= P_{k|k-1} - P_{k|k-1}C_k^T K_k^T + 0 - K_k C_k P_{k|k-1} + 0\\
&\quad + K_k C_k P_{k|k-1}C_k^T K_k^T + 0 + 0 + K_k R_v K_k^T && (11.61)\\
&= P_{k|k-1} - P_{k|k-1}C_k^T K_k^T - K_k C_k P_{k|k-1} + K_k(C_k P_{k|k-1}C_k^T + R_v)K_k^T && (11.62)
\end{aligned}
$$

Now take the derivative with respect to K_k.

$$
\begin{aligned}
\nabla_{K_k}P_{k|k} &= \nabla_{K_k}P_{k|k-1} - \nabla_{K_k}P_{k|k-1}C_k^T K_k^T - \nabla_{K_k}K_k C_k P_{k|k-1}\\
&\quad + \nabla_{K_k}K_k(C_k P_{k|k-1}C_k^T + R_v)K_k^T && (11.63)\\
0 &= 0 - P_{k|k-1}C_k^T - P_{k|k-1}C_k^T\\
&\quad + 2K_k(C_k P_{k|k-1}C_k^T + R_v) && (11.64)\\
K_k(C_k P_{k|k-1}C_k^T + R_v) &= P_{k|k-1}C_k^T && (11.65)\\
K_k &= P_{k|k-1}C_k^T(C_k P_{k|k-1}C_k^T + R_v)^{-1} && (11.66)
\end{aligned}
$$

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

$$\begin{aligned}
P_{k|k} &= P_{k|k-1} - P_{k|k-1}C_k^T K_k^T - K_k C_k P_{k|k-1} + K_k(C_k P_{k|k-1} C_k^T + R_v)K_k^T && (11.67)\\
&= P_{k|k-1} - P_{k|k-1}C_k^T (C_k P_{k|k-1} C_k^T + R_v)^{-1} C_k P_{k|k-1} \\
&\quad - P_{k|k-1}C_k^T (C_k P_{k|k-1} C_k^T + R_v)^{-1} C_k P_{k|k-1} \\
&\quad + P_{k|k-1}C_k^T (C_k P_{k|k-1} C_k^T + R_v)^{-1} (C_k P_{k|k-1} C_k^T + R_v) \\
&\quad (C_k P_{k|k-1} C_k^T + R_v)^{-1} C_k P_{k|k-1} && (11.68)\\
&= P_{k|k-1} - 2P_{k|k-1}C_k^T (C_k P_{k|k-1} C_k^T + R_v)^{-1} C_k P_{k|k-1} \\
&\quad + P_{k|k-1}C_k^T (C_k P_{k|k-1} C_k^T + R_v)^{-1} C_k P_{k|k-1} && (11.69)\\
&= P_{k|k-1} - P_{k|k-1}C_k^T (C_k P_{k|k-1} C_k^T + R_v)^{-1} C_k P_{k|k-1} && (11.70)\\
&= P_{k|k-1} - K_k C_k P_{k|k-1} && (11.71)\\
&= (I - K_k C_k) P_{k|k-1} && (11.72)
\end{aligned}$$

We thus have three equations for the prediction step

$$\begin{aligned}
K_k &= P_{k|k-1}C_k^T (C_k P_{k|k-1} C_k^T + R_v)^{-1} && (11.73)\\
\hat{x}_{k|k} &= \hat{x}_{k|k-1} + K_k(y_k - C_k \hat{x}_{k|k-1}) && (11.74)\\
P_{k|k} &= (I - K_k C_k) P_{k|k-1} && (11.75)
\end{aligned}$$

or two if you prefer

$$\begin{aligned}
\hat{x}_{k|k} &= \hat{x}_{k|k-1} + P_{k|k-1}C_k^T (C_k P_{k|k-1} C_k^T + R_v)^{-1}(y_k - C_k \hat{x}_{k|k-1}) && (11.76)\\
P_{k|k} &= P_{k|k-1} - P_{k|k-1}C_k^T (C_k P_{k|k-1} C_k^T + R_v)^{-1} C_k P_{k|k-1} && (11.77)
\end{aligned}$$

11.2.3 Putting It All Together

Predict

$$\begin{aligned}
\hat{x}_{k+1|k} &= A_k \hat{x}_{k|k} && (11.78)\\
P_{k+1|k} &= A_k P_{k|k} A_k^T + R_w && (11.79)
\end{aligned}$$

Correct

$$\begin{aligned}
K_k &= P_{k|k-1}C_k^T (C_k P_{k|k-1} C_k^T + R_v)^{-1} && (11.80)\\
\hat{x}_{k|k} &= \hat{x}_{k|k-1} + K_k(y_k - C_k \hat{x}_{k|k-1}) && (11.81)\\
P_{k|k} &= (I - K_k C_k) P_{k|k-1} && (11.82)
\end{aligned}$$

We can also combine the prediction and correction steps

Update

$$\begin{aligned}
K_k &= (A_k P_k A_k^T + R_w)C_k^T (C_k (A_k P_k A_k^T + R_w) C_k^T + R_v)^{-1} && (11.83)\\
\hat{x}_{k+1} &= A_k \hat{x}_k + K_k(y_k - C_k \hat{x}_k) && (11.84)\\
P_{k+1} &= (I - K_k C_k)(A_k P_k A_k^T + R_w) && (11.85)
\end{aligned}$$

11.3 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 $P_{k|k-1}$.

$$\begin{aligned} P_{k|k} &= U_k U_k^T && (11.86)\\ P_{k|k-1} &= S_k S_k^T && (11.87)\\ R_v^{-1} &= L_v^T L_v && (11.88)\\ R_w &= L_w L_w^T && (11.89) \end{aligned}$$

11.3.1 Prediction

The basic prediction equations are

$$\begin{aligned} \hat{x}_{k+1|k} &= A_k \hat{x}_{k|k} && (11.90)\\ P_{k+1|k} &= A_k P_{k|k} A_k^T + R_w. && (11.91) \end{aligned}$$

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

$$\begin{aligned} P_{k+1|k} &= A_k P_{k|k} A_k^T + R_w && (11.92)\\ S_{k+1} S_{k+1}^T &= A_k U_k U_k^T A_k^T + L_w L_w^T && (11.93) \end{aligned}$$

11.3.2 Correction

$$\begin{aligned} K_k &= P_{k|k-1} C_k^T (C_k P_{k|k-1} C_k^T + R_v)^{-1} && (11.94)\\ \hat{x}_{k|k} &= \hat{x}_{k|k-1} + K_k (y_k - C_k \hat{x}_{k|k-1}) && (11.95)\\ P_{k|k} &= (I - K_k C_k) P_{k|k-1} && (11.96) \end{aligned}$$

$$\begin{aligned} K_k &= S_k S_k^T C_k^T (C_k S_k S_k^T C_k^T + (L_v^T L_v)^{-1})^{-1} && (11.97)\\ \hat{x}_{k|k} &= \hat{x}_{k|k-1} + K_k (y_k - C_k \hat{x}_{k|k-1}) && (11.98)\\ U_k U_k^T &= (I - K_k C_k) S_k S_k^T && (11.99) \end{aligned}$$

Consider the last equation.

$$\begin{aligned} U_k U_k^T &= (I - S_k S_k^T C_k^T (C_k S_k S_k^T C_k^T + (L_v^T L_v)^{-1})^{-1} C_k) S_k S_k^T && (11.100)\\ &= S_k (I - S_k^T C_k^T (C_k S_k S_k^T C_k^T + (L_v^T L_v)^{-1})^{-1} C_k S_k) S_k^T && (11.101) \end{aligned}$$

Now use the matrix inversion lemma

$$U_k U_k^T = S_k(I + S_k^T C_k^T L_v^T L_v C_k S_k)^{-1} S_k^T \quad (11.102)$$

More coming...

11.4 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.

$$P_{k|k}^{-1} = U_k^T U_k \quad (11.103)$$
$$P_{k|k-1} = S_k S_k^T \quad (11.104)$$
$$R_v^{-1} = L_v^T L_v \quad (11.105)$$
$$R_w = L_w L_w^T \quad (11.106)$$

11.4.1 Correction

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

$$e_{k|k-1} = S_k \zeta_a(k) \quad (11.107)$$

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

$$e_{k|k-1} = S_k \zeta_a(k) \quad (11.108)$$
$$\hat{x}_{k|k-1} - x_k = S_k \zeta_a(k) \quad (11.109)$$
$$S_k^{-1}\hat{x}_{k|k-1} = S_k^{-1} x_k + \zeta_a(k) \quad (11.110)$$

Similarly we can write the expression for the observations

$$y_k = C_k x_k + v_k \quad (11.111)$$
$$L_v y_k = L_v C_k x_k + L_v v_k \quad (11.112)$$
$$L_v y_k = L_v C_k x_k + \zeta_b(k) \quad (11.113)$$

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

$$\begin{bmatrix} S_k^{-1}\hat{x}_{k|k-1} \\ L_v y_k \end{bmatrix} = \begin{bmatrix} S_k^{-1} \\ L_v C_k \end{bmatrix} x_k + \zeta(k) \quad (11.114)$$

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

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**.

$$Q^T \begin{bmatrix} S_k^{-1}\hat{x}_{k|k-1} \\ L_v y_k \end{bmatrix} = Q^T \begin{bmatrix} S_k^{-1} \\ L_v C_k \end{bmatrix} \hat{x}_{k|k} \tag{11.115}$$

$$= \begin{bmatrix} U_k \\ 0 \end{bmatrix} \hat{x}_{k|k} \tag{11.116}$$

Part IV

Image Processing and Machine Vision

Chapter 12

Intensity

12.1 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

$$N(x, y) \quad = \quad (n-1) - I(x, y) \tag{12.1}$$

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).

12.2 Convolution Masks

12.3 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).

12.3.1 Differentiation

$$derivative = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \tag{12.2}$$

Sobel

$$S_0 = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \tag{12.3}$$

$$S_1 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{12.4}$$

12.3.2 High Pass Filters

$$Laplacian1 = \begin{bmatrix} -1 & 0 & -1 \\ 0 & 4 & 0 \\ -1 & 0 & -1 \end{bmatrix} \tag{12.5}$$

$$Laplacian2 = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \tag{12.6}$$

$$sharpen = \begin{bmatrix} 0 & -1 & 0 \\ -1 & a & -1 \\ 0 & -1 & 0 \end{bmatrix} \tag{12.7}$$

$$a \in \{5, 6, 7\} \tag{12.8}$$

Chapter 13

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.

13.1 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 12.1.

Listing 13.1: 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 12.2. Note we have included a binary search version, which is faster since the bins are ordered.

Listing 13.2: 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
```

13.2 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

$$|pdf_B dB| = |pdf_A dA| \tag{13.1}$$

$$pdf_B = pdf_A \left|\frac{dA}{dB}\right| \tag{13.2}$$

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

$$
\begin{aligned}
pdf_B &= pdf_A \left| \left(\frac{dB}{dA} \right)^{-1} \right| && (13.3)\\
&= pdf_A \left| \left(\frac{dT(A)}{dA} \right)^{-1} \right| && (13.4)\\
&= pdf_A \left| \left(\frac{d \int_{-\infty} apdf_A dA}{dA} \right)^{-1} \right| && (13.5)\\
&= pdf_A \left| (pdf_A)^{-1} \right| && (13.6)\\
&= \frac{pdf_A}{pdf_A} && (13.7)\\
&= 1 && (13.8)
\end{aligned}
$$

Thus the pdf of B is uniform on the interval 0 to 1, and is thus white-balanced. The code in Listing **??** 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 13.3: 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.

13.3 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.

13.3.1 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 13.4: 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
```

13.3.2 SciLab Implementation

Consider code 12.3.

Listing 13.5: 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 12.1. 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)

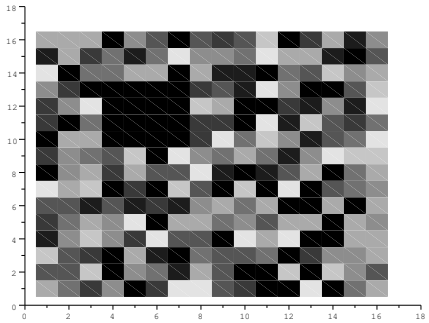
```
A=rand(20,20);
A(4:7,4:7)=ones(4,4);
```

The specific commands to generate each picture were

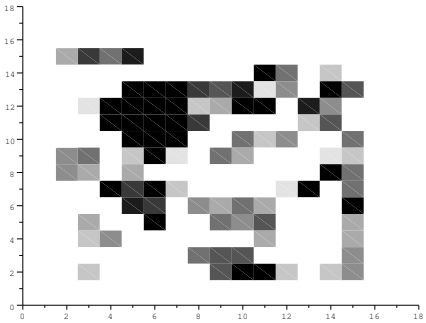
```
f=scf();
Matplot(10-A.*hist_mask(A,threshold=0.3).*10);
f.colormap=graycolormap(10);
```

Table 13.1: Histogram thresholds to yield regions of interest.

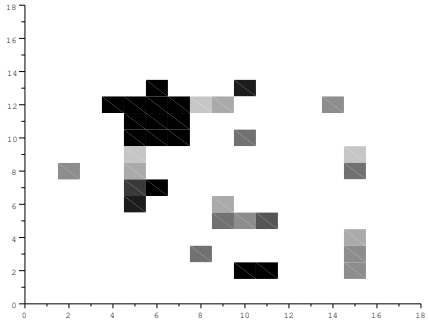
Original



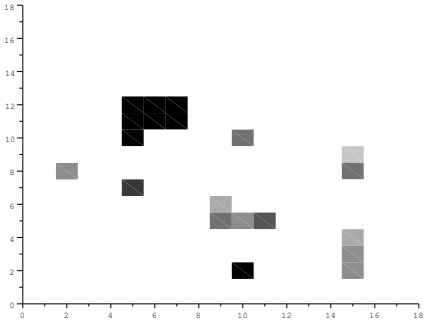
Threshold =0.1



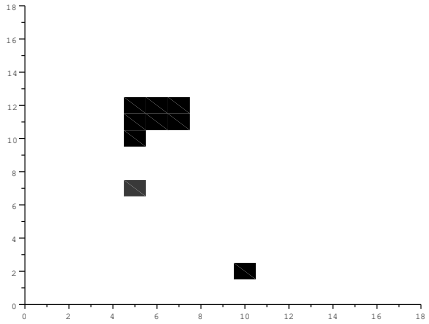
Threshold = 0.3



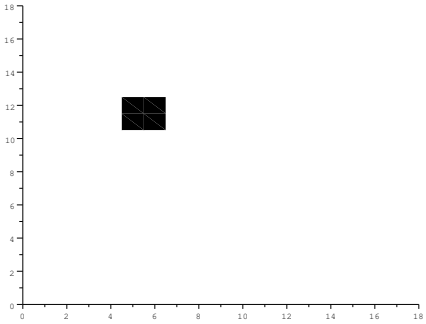
Threshold =0.4



Threshold = 0.5



Threshold =0.6



Chapter 14

Filtering

14.1 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,

$$y(k,l) = h(k,l) * x(k,l) \tag{14.1}$$

or in the frequency domain,

$$Y(u,v) = H(u,v)X(u,v). \tag{14.2}$$

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,

$$\hat{X}(u,v) = H^{-1}(u,v)Y(u,v). \tag{14.3}$$

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

$$\begin{aligned} \hat{X}(u,v) &= H^{-1}(u,v)Y(u,v) && (14.4)\\ &= H^{-1}(u,v)H(u,v)X(u,v) && (14.5)\\ &= X(u,v). && (14.6) \end{aligned}$$

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

$$\begin{aligned} \hat{X}(u,v) &= H^{-1}(u,v)Y(u,v) && (14.7)\\ &= H^{-1}(u,v)\left(H(u,v)X(u,v) + N(u,v)\right) && (14.8)\\ &= X(u,v) + H^{-1}(u,v)N(u,v). && (14.9) \end{aligned}$$

[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.

14.2 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

$$mse(x,\hat{x}) = E\left[(x-\hat{x})^2\right] \tag{14.10}$$

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

$$\begin{aligned}
MSE(X,\hat{X}) &= E\left[(X-\hat{X})^2\right] && (14.11)\\
&= E\left[(X-GY)^2\right] && (14.12)\\
&= E\left[(X-G(HX+N))^2\right] && (14.13)\\
&= E\left[(X-GHX-GN)^2\right] && (14.14)\\
&= E\left[((I-GH)X-GN)^2\right] && (14.15)\\
&= E\left[((I-GH)X-GN)^*((I-GH)X-GN)\right] && (14.16)\\
&= E\left[(X^*(I-GH)^*-N^*G^*)((I-GH)X-GN)\right] && (14.17)\\
&= E\left[X^*(I-GH)^*((I-GH)X-GN)-N^*G^*((I-GH)X-GN)\right] && (14.18)\\
&= E\left[X^*(I-GH)^*(I-GH)X-X^*(I-GH)^*GN-N^*G^*(I-GH)X+N^*G^*GN\right] && (14.19)\\
&= E\left[X^*(I-GH)^*(I-GH)X+N^*G^*GN\right] && (14.20)\\
&= (I-GH)^*(I-GH)S_X+G^*GS_N && (14.21)\\
&= (I-H^*G^*)(I-GH)S_X+G^*GS_N && (14.22)\\
&= (I-H^*G^*-GH+H^*G^*GH)S_X+G^*GS_N && (14.23)
\end{aligned}$$

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

$$\begin{aligned}
\min_G MSE(X,\hat{X}) &= \nabla_G\left((I-H^*G^*-GH+H^*G^*GH)S_X+G^*GS_N\right) && (14.24)\\
0 &= -2H^*S_X+2GHH^*S_X+2GS_N && (14.25)\\
2H^*S_X &= 2G(HH^*S_X+S_N) && (14.26)\\
H^* &= G(HH^*+\frac{S_N}{S_X}) && (14.27)\\
G &= \frac{H^*}{(HH^*+\frac{S_N}{S_X})} && (14.28)
\end{aligned}$$

Table 14.1: 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,

$$G = \frac{H^*}{(HH^* + \frac{S_N}{S_X})} \tag{14.29}$$

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.

14.3 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 14.1: 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 14.1: Wiener filtering Images.

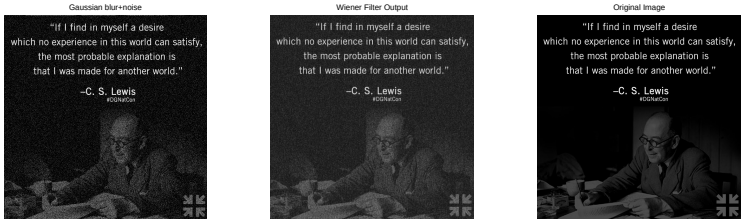


Figure 14.2: Zoomed view of Wiener filtering Images.

14.4 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

$$\|g - Hf\|_2 \quad = \quad \|\eta\|_2 \tag{14.30}$$

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

$$\hat{f} \quad = \quad \text{argmin}_{\,f} \|Cf\|_2^2 \tag{14.31}$$

$$s.t.\ \|g - Hf\|_2^2 = \|\eta\|^2 \tag{14.32}$$

Chapter 15

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 **??**.

15.1 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 **??**. 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 15.1: The 3D CT image is composed of a series of stacked cut planes.

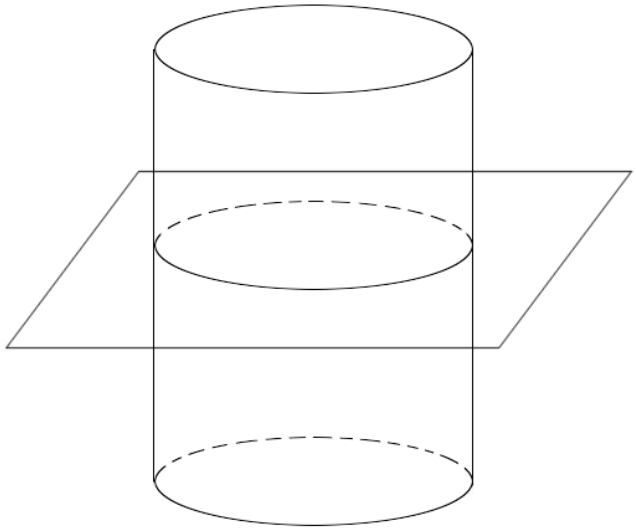


Figure 15.2: The Radon transform deals with a detector plane at angle θ to the horizontal, and a projection at some distance R from a parallel projection that passes through the origin of the system.

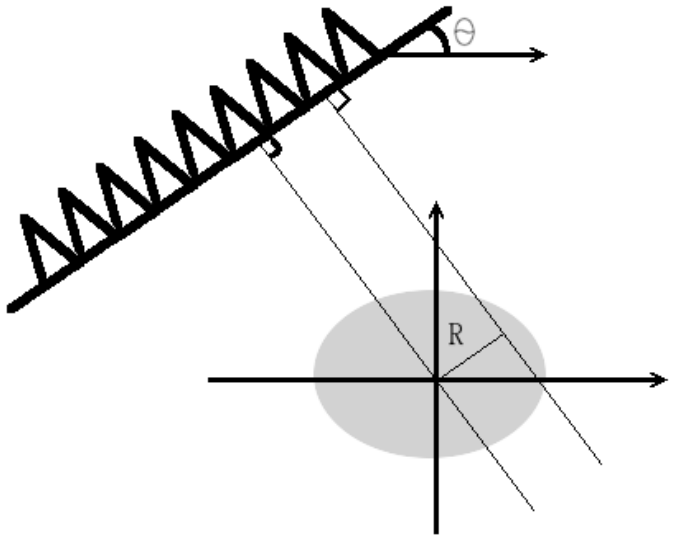
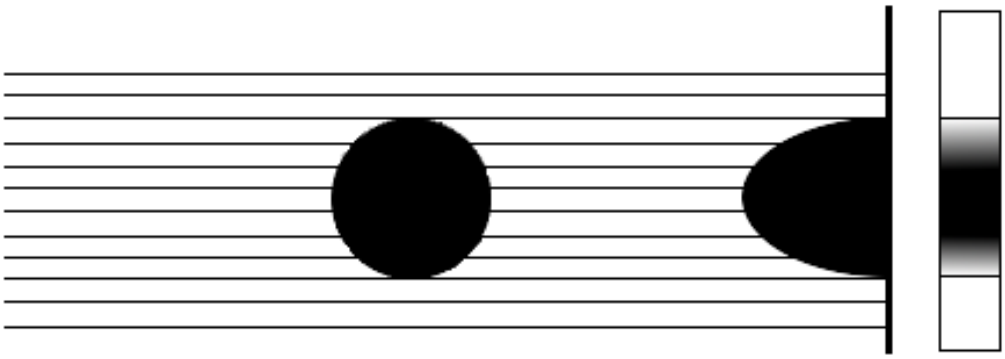


Figure 15.3: One object projected onto a plane, showing the measured attenuation/losses. These could be energy loses, photon loses, etc.



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

15.2 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 **??**, it is easy to know the path, and thus easier to reconstruct the image.

Figure 15.4: One object projected onto a series of planes in a CT scan.

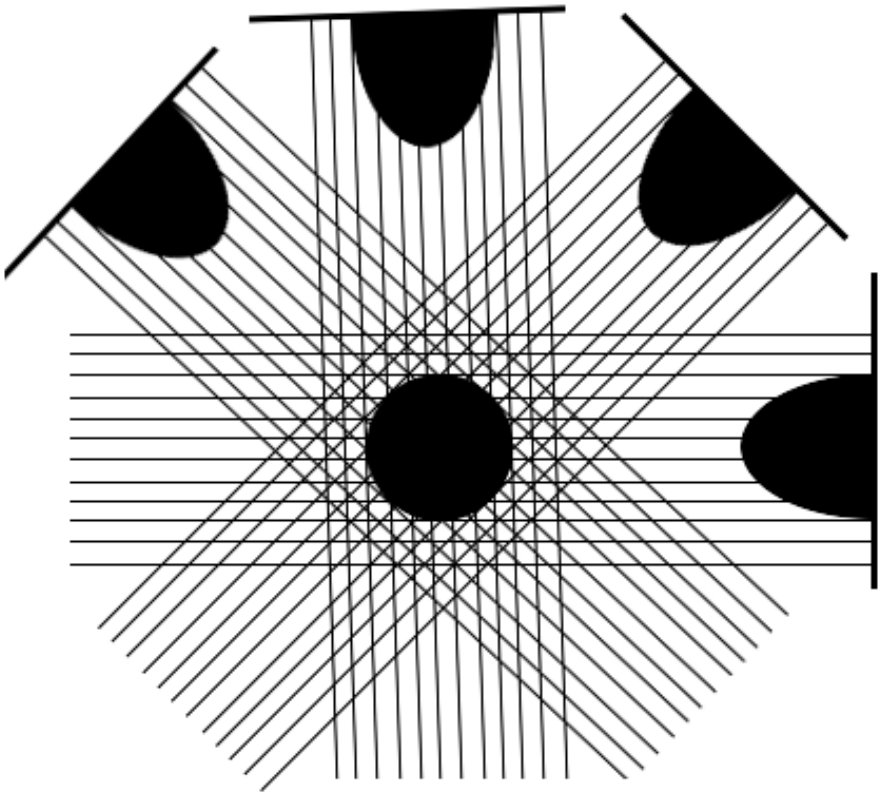


Figure 15.5: Two objects projected onto a series of planes in a CT scan.

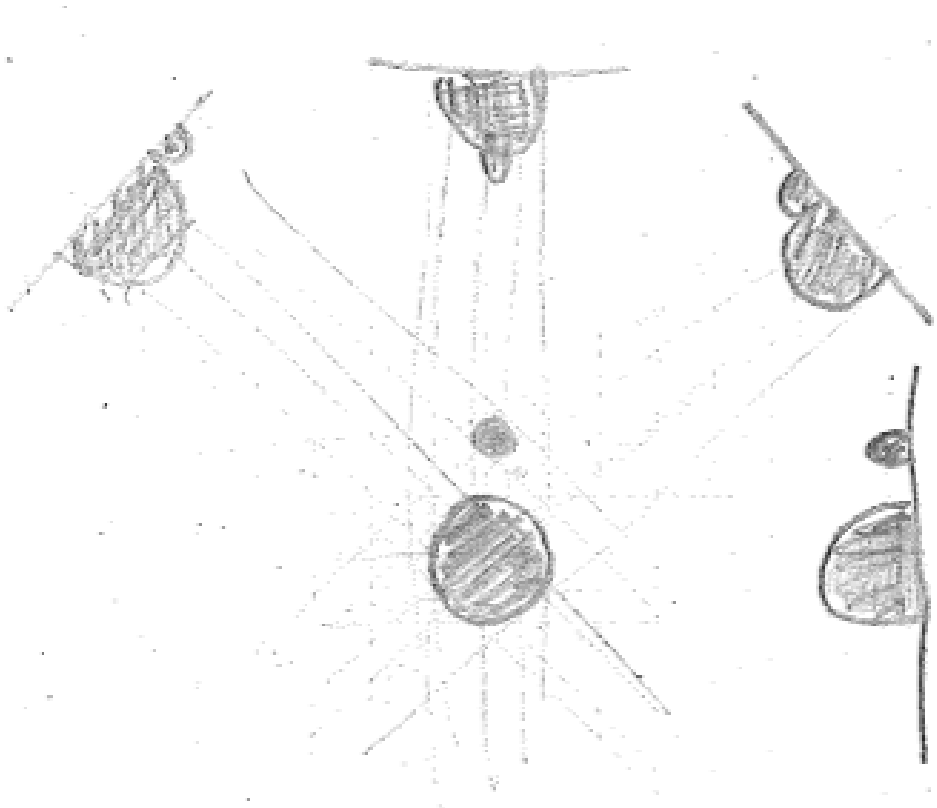


Figure 15.6: Sinogram for the two objects in Figure **??**.

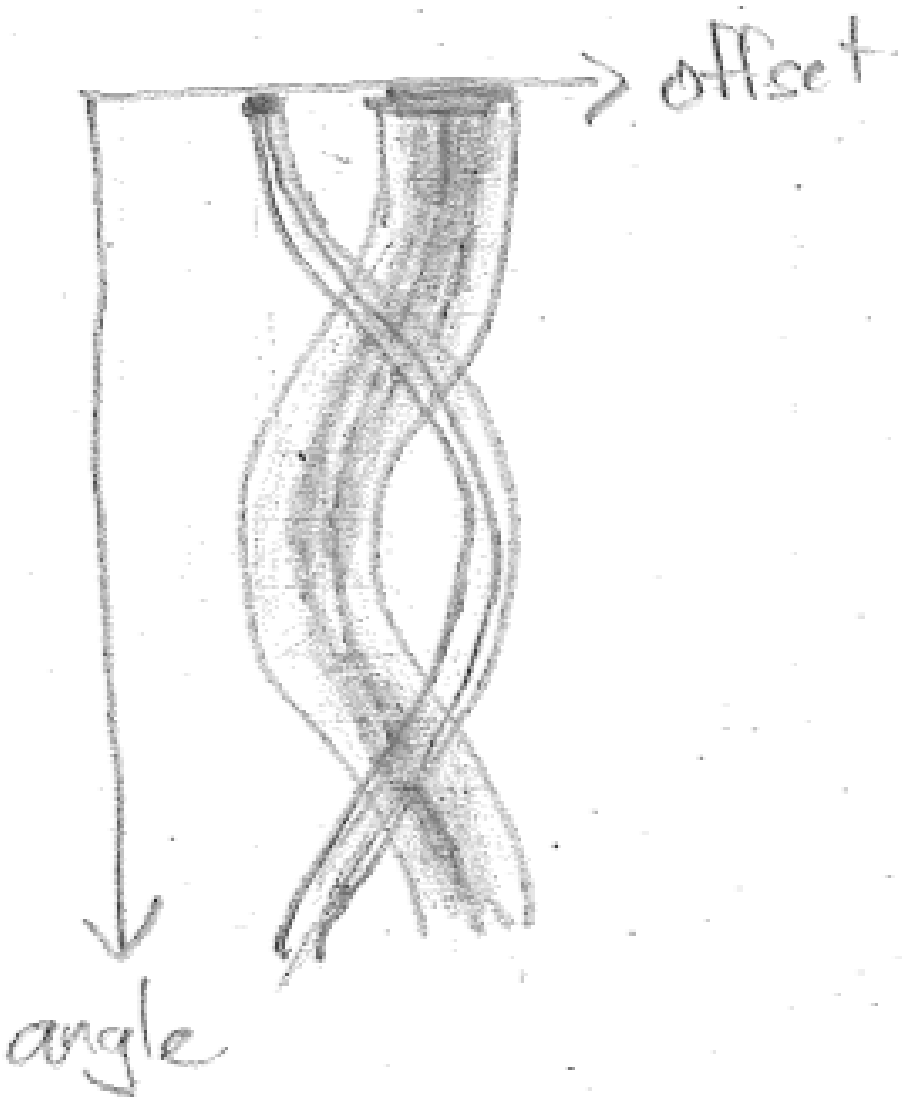
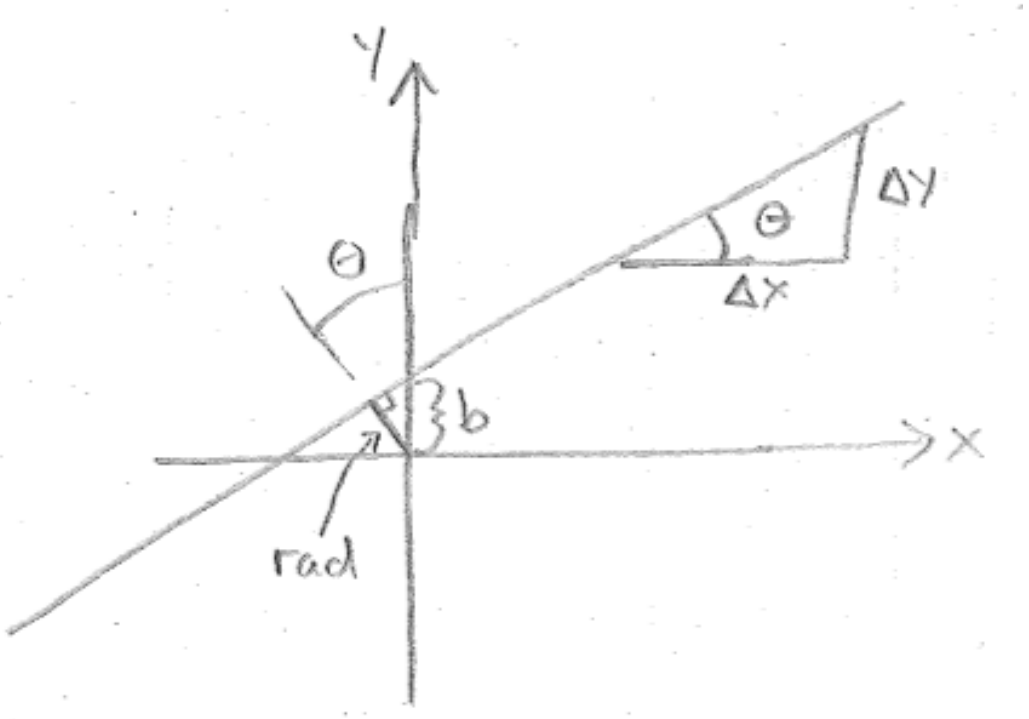


Figure 15.7: Projection line at rotational angle θ.



15.3 Fan Beam CT

15.4 Back Projection

15.5 Filtered Back Projection

15.6 Algebraic Reconstruction Technique

$$x_{element}(k+1) = x_{element}(k) + \frac{A_{projection,element} r_{projection}(k)}{A_{projection,*}} \quad (15.1)$$

$$A_{row,*} = \sum_{col=1}^{no\ of\ cols} A_{row,col} \quad (15.2)$$

$$r_{row}(k) = b_{row} - \sum_{col=1}^{no\ of\ cols} A_(row,col) x_{col}(k) \quad (15.3)$$

$$(15.4)$$

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.

15.7 Simultaneous Algebraic Reconstruction Technique

$$x_{element}(k+1) = x_{element}(k) + \frac{\omega}{A_{*,element}} \sum_{row=1}^{no\ of\ rows} \frac{A_{row,element} r_{row}(k)}{A_{row,*}} \quad (15.5)$$

$$A_{*,element} = \sum_{row=1}^{no\ of\ rows} A_{row,element} \quad (15.6)$$

$$A_{row,*} = \sum_{col=1}^{no\ of\ cols} A_{row,col} \quad (15.7)$$

$$r_{row}(k) = b_{row} - \sum_{col=1}^{no\ of\ cols} A_(row,col) x_{col}(k) \quad (15.8)$$

$$\omega \in [0,2] \quad (15.9)$$

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.

15.8 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.)

15.9 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. **??**. 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

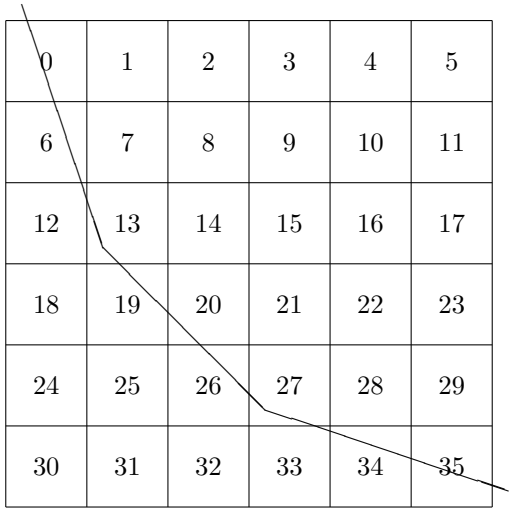


Figure 15.8: Trajectory of first particle.

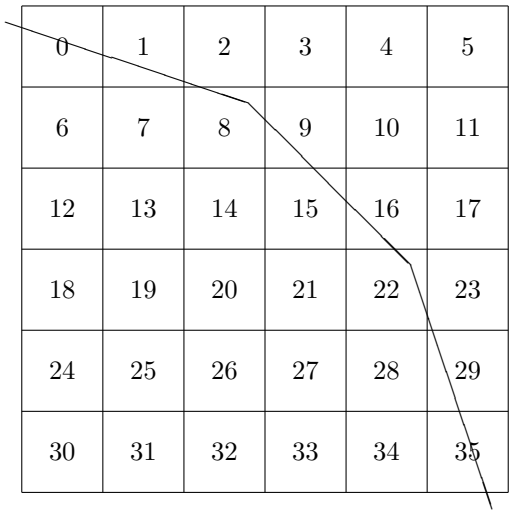


Figure 15.9: Trajectory of second particle.

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. **??**, 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. **??**. 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 16

Removing Distortion From MR Images

16.1 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

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

16.2 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).

$$P_2 = P_1 + hN \tag{16.1}$$

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

$$p^T N = d_2 \tag{16.2}$$

Using both Eq **??** and Eq **??** we obtain

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

16.3 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 $\bar{z}$.

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

Part V

Appendices

Appendix A

Quaternion

A quaternion is a four dimensional scalar, that has three different roots of negative one, i, j, and k.

$$i^2 = j^2 = k^2 = ijk = -1 \tag{A.1}$$

This results in the following useful results

- $ij = k$
- $jk = i$
- $ki = j$
- $ji = -k$
- $kj = -i$
- $ik = -j$

Appendix B

Matrix Preliminaries

Matrices are an essential tool in control systems design. First lets discuss some basic terms.

$\mathbb{R}$ The real numbers.

$\mathbb{R}^n$ The vectors composed of n-tuples of real numbers.

$\mathbb{R}^{m\times n}$ The matrices composed of m rows and n columns of real numbers.

B.1 Addition and Subtraction

In order to add or subtract matrices, the dimensions must be the same. Essentially we can add two $\mathbb{R}^{m\times n}$ matrices but not a $\mathbb{R}^{m\times n}$ and a $\mathbb{R}^{p\times r}$ matrix or even a $\mathbb{R}^{m\times n}$ and a $\mathbb{R}^{m\times m}$ matrix. Addition (or subtraction) is done by adding (or subtracting) the corresponding elements in the two matrices. For two $\mathbb{R}^{3\times 2}$ matrices addition is given by

$$\begin{bmatrix}1 & 2\\ 3 & 4\\ 5 & 6\end{bmatrix}+\begin{bmatrix}7 & 8\\ 9 & 10\\ 11 & 12\end{bmatrix}=\begin{bmatrix}1+7 & 2+8\\ 3+9 & 4+10\\ 5+11 & 6+12\end{bmatrix}=\begin{bmatrix}8 & 10\\ 12 & 14\\ 16 & 18\end{bmatrix} \tag{B.1}$$

For two $\mathbb{R}^{3\times 2}$ matrices subtraction is given by

$$\begin{bmatrix}1 & 2\\ 3 & 4\\ 5 & 6\end{bmatrix}-\begin{bmatrix}6 & 5\\ 4 & 3\\ 2 & 1\end{bmatrix}=\begin{bmatrix}1-6 & 2-5\\ 3-4 & 4-3\\ 5-2 & 6-1\end{bmatrix}=\begin{bmatrix}-5 & -3\\ -1 & 1\\ 3 & 5\end{bmatrix} \tag{B.2}$$

B.2 Multiplication

In order to do multiplication, we need to have matrices that are compatible to multiply. Recall that in order to add two matrices they had to be the same size. Unfortunately this is not the condition for multiplication. To be able to multiply a matrix A on the right by a

matrix B[1], we must have that the inner matrix dimensions are equal. That is, we require that $A \in \mathbb{R}^{m\times p}$ and $B \in \mathbb{R}^{p\times n}$, Where m, n, and p do not have to be the same, but they could. Thus we could multiply the following

- AB or BA with $\{A, B\} \in \mathbb{R}^{3\times 3}$;
- AB with $A \in \mathbb{R}^{3\times 3}$ and $B \in \mathbb{R}^{3\times 4}$;
- AB with $A \in \mathbb{R}^{2\times 3}$ and $B \in \mathbb{R}^{3\times 4}$.

Two general ways to do multiplication exist. Each has computational advantages in different situations. They give the same answer they are just different ways to group the solution.

B.2.1 Inner Product

A full study of the inner product is beyond the scope of this work, but interested students are referred to texts on Hilbert Spaces[2]. An inner product of two vectors of size n[3] a and b is given by $\langle a, b\rangle = a^T b = \sum_{i=1}^{n}(a_i b_i)$. Thus it is the sum of the product of corresponding elements in the vectors.

Example:

$$a = \begin{bmatrix}1 & 2 & 3\end{bmatrix}^T \qquad b = \begin{bmatrix}4 & 5 & 6\end{bmatrix}^T$$

Note that I have defined the vectors in the transposed form to save space. The transpose just means

$$\begin{bmatrix}1 & 2\end{bmatrix}^T = \begin{bmatrix}1\\2\end{bmatrix}. \tag{B.3}$$

Then the inner product of a and b is

$$\begin{aligned}\langle a, b\rangle &= \begin{bmatrix}1 & 2 & 3\end{bmatrix}\begin{bmatrix}4\\5\\6\end{bmatrix}\\ &= 1\times 4 + 2\times 5 + 3\times 6\\ &= 32\end{aligned}$$

Now try it in SciLab. First define a and b by

```
--> a=[1;2;3];
--> b=[4;5;6];
```

[1] Multiplying a matrix A on the right by a matrix B means AB, left multiplication by B would be BA. In matrices it is not true in general that $AB = BA$, or even that one can be calculated even if the other exists. This will make more sense in a few seconds.

[2] Hilbert Spaces are spaces with a defined inner product. They play an important role in control systems theory

[3] A vector of size n is the same as saying the vector is in $\mathbb{R}^n$.

Note that I ended each line with a ";", which tells SciLab to suppress its responses. This is vital when using the programming mode or SciLab will scroll out lots of unneeded info. With the vectors described we just need to take the inner product. To do this we use the $a^T b$ form, which is written

```
--> a'*b
```

The prime(') does the transpose (^{T}) and the multiply then does the inner product.

With vector inner products down we can consider two matrices, $A \in \mathbb{R}^{4\times 2}$ and $B \in \mathbb{R}^{2\times 3}$.

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \\ a_{3,1} & a_{3,2} \\ a_{4,1} & a_{4,2} \end{bmatrix} \qquad B = \begin{bmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,1} & b_{2,2} & b_{2,3} \end{bmatrix}$$

We will consider partitioning the matrices into vectors that can be multiplied. We define

- The first row of A to be $a_1 = \begin{bmatrix} a_{1,1} & a_{1,2} \end{bmatrix}$.
- The second row of A to be $a_2 = \begin{bmatrix} a_{2,1} & a_{2,2} \end{bmatrix}$.
- The third row of A to be $a_3 = \begin{bmatrix} a_{3,1} & a_{3,2} \end{bmatrix}$.
- The fourth row of A to be $a_4 = \begin{bmatrix} a_{4,1} & a_{4,2} \end{bmatrix}$.
- The first column of B to be $b_1 = \begin{bmatrix} b_{1,1} & b_{2,1} \end{bmatrix}^T$.
- The second column of B to be $b_2 = \begin{bmatrix} b_{1,2} & b_{2,2} \end{bmatrix}^T$.
- The third column of B to be $b_3 = \begin{bmatrix} b_{1,3} & b_{2,3} \end{bmatrix}^T$.

then

$$A = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} \qquad B = \begin{bmatrix} b_1 & b_2 & b_3 \end{bmatrix} \tag{B.4}$$

and so the product

$$\begin{aligned} AB &= \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} \begin{bmatrix} b_1 & b_2 & b_3 \end{bmatrix} \\ &= \begin{bmatrix} a_1 b_1 & a_1 b_2 & a_1 b_3 \\ a_2 b_1 & a_2 b_2 & a_2 b_3 \\ a_3 b_1 & a_3 b_2 & a_3 b_3 \\ a_4 b_1 & a_4 b_2 & a_4 b_3 \end{bmatrix} \\ &= \begin{bmatrix} \sum_{i=1}^{2} a_{1,i} b_{i,1} & \sum_{i=1}^{2} a_{1,i} b_{i,2} & \sum_{i=1}^{2} a_{1,i} b_{i,3} \\ \sum_{i=1}^{2} a_{2,i} b_{i,1} & \sum_{i=1}^{2} a_{2,i} b_{i,2} & \sum_{i=1}^{2} a_{2,i} b_{i,3} \\ \sum_{i=1}^{2} a_{3,i} b_{i,1} & \sum_{i=1}^{2} a_{3,i} b_{i,2} & \sum_{i=1}^{2} a_{3,i} b_{i,3} \\ \sum_{i=1}^{2} a_{4,i} b_{i,1} & \sum_{i=1}^{2} a_{4,i} b_{i,2} & \sum_{i=1}^{2} a_{4,i} b_{i,3} \end{bmatrix} \end{aligned}$$

By hand the easiest way to do this is to line up the two matrices to multiply as follows

$$\begin{bmatrix} & & \\ & & \\ & & \\ & & \\ \end{bmatrix} \begin{bmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,1} & b_{2,2} & b_{2,3} \end{bmatrix}$$
$$\begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \\ a_{3,1} & a_{3,2} \\ a_{4,1} & a_{4,2} \end{bmatrix} \begin{bmatrix} \square & \square & \square \\ \square & \square & \square \\ \square & \square & \square \\ \square & \square & \square \end{bmatrix}$$

Then calculate each component by lining up the row and column and multiplying the corresponding terms in them.

$$\begin{bmatrix} & & \\ & & \\ & & \\ & & \\ \end{bmatrix} \begin{bmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,1} & b_{2,2} & b_{2,3} \end{bmatrix}$$
$$\begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \\ a_{3,1} & a_{3,2} \\ a_{4,1} & a_{4,2} \end{bmatrix} \begin{bmatrix} a_{1,1}b_{1,1} + a_{1,2}b_{2,1} & \square & \square \\ \square & \square & \square \\ \square & \square & \square \\ \square & \square & \square \end{bmatrix}$$

The second element is

$$\begin{bmatrix} & & \\ & & \\ & & \\ & & \\ \end{bmatrix} \begin{bmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,1} & b_{2,2} & b_{2,3} \end{bmatrix}$$
$$\begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \\ a_{3,1} & a_{3,2} \\ a_{4,1} & a_{4,2} \end{bmatrix} \begin{bmatrix} a_{1,1}b_{1,1} + a_{1,2}b_{2,1} & \square & \square \\ a_{2,1}b_{1,1} + a_{2,2}b_{2,1} & \square & \square \\ \square & \square & \square \\ \square & \square & \square \end{bmatrix}$$

The third element is

$$\begin{bmatrix} & & \\ & & \\ & & \\ & & \\ \end{bmatrix} \begin{bmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,1} & b_{2,2} & b_{2,3} \end{bmatrix}$$
$$\begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \\ a_{3,1} & a_{3,2} \\ a_{4,1} & a_{4,2} \end{bmatrix} \begin{bmatrix} a_{1,1}b_{1,1} + a_{1,2}b_{2,1} & \square & \square \\ a_{2,1}b_{1,1} + a_{2,2}b_{2,1} & \square & \square \\ a_{3,1}b_{1,1} + a_{3,2}b_{2,1} & \square & \square \\ \square & \square & \square \end{bmatrix}$$

The fourth element is

$$\begin{bmatrix} & & \\ & & \\ & & \\ & & \\ \end{bmatrix} \begin{bmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,1} & b_{2,2} & b_{2,3} \end{bmatrix}$$
$$\begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \\ a_{3,1} & a_{3,2} \\ a_{4,1} & a_{4,2} \end{bmatrix} \begin{bmatrix} a_{1,1}b_{1,1} + a_{1,2}b_{2,1} & \square & \square \\ a_{2,1}b_{1,1} + a_{2,2}b_{2,1} & \square & \square \\ a_{3,1}b_{1,1} + a_{3,2}b_{2,1} & \square & \square \\ a_{4,1}b_{1,1} + a_{4,2}b_{2,1} & \square & \square \end{bmatrix}$$

Then repeat the process till every box is filled.

B.3 Matrix Classification

B.3.1 Basic Properties

Property	Meaning
Diagonal	$A_{ij} = 0 \forall i \neq j$
Lower Triangular	$A_{ij} = 0 \forall i < j$
Upper Triangular	$A_{ij} = 0 \forall i > j$
Tridiagonal	$A_{ij} = 0 \forall \lvert i-j \rvert \leq 1$
Pentediagonal	$A_{ij} = 0 \forall \lvert i-j \rvert \leq 3$
Lower Hessenberg	$A_{ij} = 0 \forall i+1 > j$
Upper Hessenberg	$A_{ij} = 0 \forall i > j+1$
Symmetric	$A = A^T$
Normal	$AA^H = A^H A$
Idempotent	$A^2 = A$
Orthogonal	$A^{-1} = A^T$
Unitary	$A^{-1} = A^H$
Positive Definite ($A > 0$)	$x^H Ax > 0 \forall x \in \mathbb{C}^n$

B.3.2 Definite

Positive Definite

A matrix, $A \in \mathbb{C}^{n \times n}$ is defined as positive definite if $\forall x \in \mathbb{C}^n$, $x^H Ax > 0$. Usually we further restrict this to symmetric matrices, as they are what comes up in practice and they are easier to handle (more algorithms, easier proofs, etc.), but they are not the only matrices which fit the definition[4].

Positive definite matrices are non-singular, and thus invertible. In some sense they are the nicest matrices to work with.

Theorem 2 *Let A be any symmetric positive definite matrix of size n, with smallest singular value σ_n. Let q_1 and q_2 be two orthonormal n-vectors and $1 > \alpha > 0$ be a scalar. The matrix $A + n\alpha\sigma_n q_1 q_2^H$ is non-symmetric positive definite.*

Proof:

Let Q be an orthogonal matrix with the first two columns q_1 and q_2 respectively. Q forms a basis for the n-vectors, so $\forall y \in \mathbb{C}^n \exists x \in \mathbb{C}^n$, $y = Qx$. Since Q is unitary it is invertible, so it is 1-1 and onto, thus every x has a unique y and vice versa. Let the singular value decomposition of A be $U\Sigma U^H$. Now consider

$$\begin{aligned} y^H(A + n\alpha\sigma_n q_1 q_2^H)y &= y^H Ay + n\alpha\sigma_n y^H q_1 q_2^H y && \text{(B.5)} \\ &= x^H Q^H AQx + n\alpha\sigma_n x^H Q^H q_1 q_2^H Qx && \text{(B.6)} \\ &= x^H Q^H U\Sigma U^H Qx + n\alpha\sigma_n x_1 x_2 && \text{(B.7)} \\ &= x^H V^H \Sigma V x + n\alpha\sigma_n x_1 x_2 && \text{(B.8)} \end{aligned}$$

[4]It is worth noting that there are non-symmetric positive definite matrices. I have never seen anyone categorize how to create one, so I decided to do it in Theorem 2 on page 85.

for $V = U^H Q$. Note that $V^H \Sigma V$ must be symmetric positive definite, thus $x^H V^H \Sigma V x \geq \|x\|_2^2 \sigma_n$. If x_1 or x_2 are zero then it is trivial that $x^H V^H \Sigma V x + n\alpha\sigma_n x_1 x_2 > 0$, so it only remains to prove positive definiteness when both are not zero. That means

$$\begin{aligned} y^H(A + n\alpha\sigma_n q_1 q_2^H)y &\geq \|x\|_2^2\sigma_n + n\alpha\sigma_n x_1 x_2 && \text{(B.9)} \\ &\geq n\|x\|_\infty^2\sigma_n + n\alpha\sigma_n x_1 x_2 && \text{(B.10)} \\ &\geq n|x_1 x_2|\sigma_n + n\alpha\sigma_n x_1 x_2 && \text{(B.11)} \end{aligned}$$

Now factor this expression as follows

$$\begin{aligned} n|x_1||x_2|\sigma_n + n\alpha\sigma_n x_1 x_2 &= n|x_1||x_2|\sigma_n + n\alpha\sigma_n |x_1||x_2| sign(x_1 x_2) && \text{(B.12)} \\ &= (1 + \alpha sign(x_1 x_2))n|x_1||x_2|\sigma_n. && \text{(B.13)} \end{aligned}$$

Trivially, $n|x_1||x_2|\sigma_n > 0$, and $1+\alpha sign(x_1 x_2) > 0$, so $n|x_1||x_2|\sigma_n + n\alpha\sigma_n x_1 x_2 > 0$ and thus we have $y^H(A + n\alpha\sigma_n q_1 q_2^H)y > 0$. Since this holds for all y, we have that $A + n\alpha\sigma_n q_1 q_2^H$ is positive definite. The non-symmetry is a direct result of the structure.

$\diamondsuit$ SDG $\diamondsuit$

Appendix C

Using SciLab

C.1 Basics

When you first start SciLab you will see something like

```
                ==========
                 scilab-2.7.2
        Copyright (C) 1989-2003 INRIA/ENPC
                ==========



Startup execution:
  loading initial environment

-->
```

The arrow "−−>" is the command prompt. SciLab, like MatLab is a command line interface to a mathematics programming environment. To get started lets do a calculation.

```
3+(2+5*4)/11
```

SciLab performs the calculation and displays the answer.

```
 ans  =

    5.
```

Now lets define a simple variable.

```
a=2
```

SciLab responds with

```
 a  =

    2.
```

Notice anything similar? The response is almost the same but "ans" has been replaced by the variable name "a". In fact it is even more similar than that. When no assignment ("name=") is given, SciLab automatically assigns the result to the variable "ans". Try using it.

```
a*ans
```

SciLab will tell you that "ans" is now 10. Lets move on and define a matrix. Type the following

```
  A=[1,2;3,4;5,6]
```

and press enter. Commas are used to separate elements and semicolons are used to separate rows. Note that you could also have entered "A" using the alternate notation

```
  A=[[1 2];[3 4];[5 6]]
```

or even (command prompt shown so you won't think something is wrong when it automatically appears, also you do not need to space over like I do to enter the numbers, I just find it easier to read)

```
-->  A=[[1 2]
-->     [3 4]
-->     [5 6]]
```

Thus spaces work like commas and returns work like semicolons. In any case, SciLab should respond by showing you that it has created the matrix variable as follows

```
 A  =

!   1.    2. !
!   3.    4. !
!   5.    6. !
```

The variable "A" is now defined and can be used. For instance we might want to define "B" to be "A+A". Do this by typing

```
  B=A+A
```

SciLab will add the matrices and define "B" to be the result, showing you the answer.

```
 B  =

!   2.     4.  !
!   6.     8.  !
!   10.    12. !
```

This mode is useful for doing simple calculations and testing output. We will refer to it as the interactive mode. Since SciLab has an interactive mode that is command driven, it is reasonable to assume it would have a programming interface (we will refer to it as the programming mode). I will show the use of programming mode later.

C.2 Scilab and Matlab Programming

Scilab is a Matlab look alike. We will use Scilab as it is free and open source, but it is useful to also know about Matlab.

C.2.1 Matlab

There are two basic ways to interact with Matlab: command line execution, and M-files. Yes there are others such as MEX-files, Simulink, and several interfacing programs, but they are not relevant to us. We will primarily be concerned with the use of M-files, because they are the most helpful. Command line execution is really just for quick operations and checking of segments of code. Matlab syntax is a high level programming language that interacts with a series of numerical libraries (most notably LinPack, EisPack, and BLAS). Like most programming languages we have two types of programs that can be written. A regular program, which is written as you would type commands on the command line, is the most basic type and is often the way you will start homework problems and other projects. Functions, which are sub-programs called by another program (even recursively by other functions), are probably the most useful, as they allow you to extend the language by defining new operations. One of the main goals of this class is for you to walk away with a library of Matlab functions that you can use to do a variety of tasks. So how do you specify which you want? You will get a regular program unless you start the M-file with the command function. The syntax is

function a=name(x,y,..., z)

or

function [a,b,...,c]=name(x,y,..., z)

The second form returns multiple values. Matlab gives us several command structures also: for, while, and if-elseif-else. To see how these work let's use the programs I passed out last time as an example.

Homework: Convert the Fortran program in 3.1 into Matlab syntax. Do problems 9, 13, 14 from section 3.1