Keith On . . .
Robotics and Control

K.E. Schubert

Founder
Renaissance Research Labs

Professor
Department of Computer Science and Engineering
California State University, San Bernardino

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).

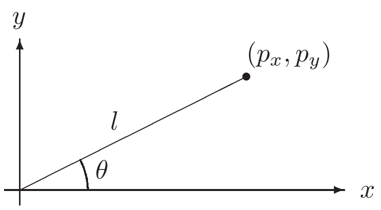
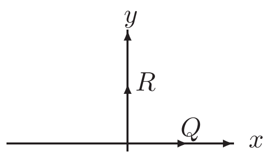
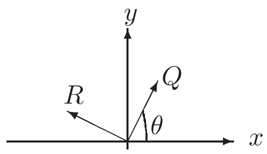
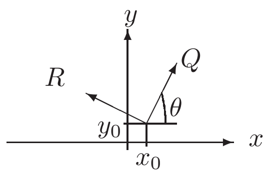


Figure 1.1: Single Radial Arm

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

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.

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

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

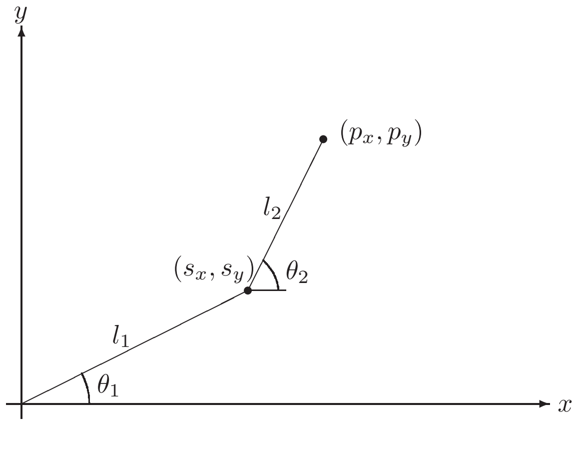


Figure 1.2: Arm With Two Radial Joints

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

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

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.

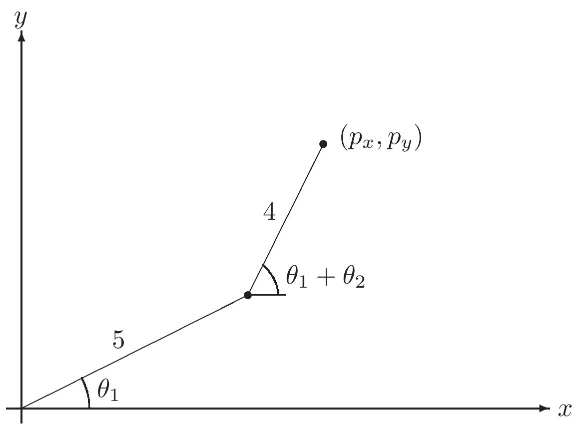


Figure 2.1: Arm With Two Radial Joints

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.

$$\mathfrak{L} = \mathfrak{K} - \mathfrak{P} \tag{4.10}$$

$$\sum F = \frac{d}{dt}\frac{\partial \mathfrak{L}}{\partial \dot{x}} - \frac{\partial \mathfrak{L}}{\partial x} \tag{4.11}$$

$$= 0 \tag{4.12}$$

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.

$$\sum T = \frac{d}{dt}\frac{\partial \mathfrak{L}}{\partial \dot{\theta}} - \frac{\partial \mathfrak{L}}{\partial \theta} \tag{4.13}$$

$$= 0 \tag{4.14}$$

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:*

$$K = \frac{1}{2}I\dot{\theta}^2. \tag{4.24}$$

[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.

$$K_x = \frac{1}{2}\int_0^l \frac{m}{l} r^2 dr \dot{\theta}^2 sin^2(\theta) \tag{4.15}$$

$$= \frac{1}{2}\frac{m}{l}\int_0^l r^2 dr \dot{\theta}^2 sin^2(\theta) \tag{4.16}$$

$$= \frac{1}{2}\frac{m}{l}\left.\frac{r^3}{3}\right|_0^l \dot{\theta}^2 sin^2(\theta) \tag{4.17}$$

$$= \frac{1}{2}\frac{m}{l}\left(\frac{l^3}{3}\right)\dot{\theta}^2 sin^2(\theta) \tag{4.18}$$

$$= \frac{1}{2}m\left(\frac{l^2}{3}\right)\dot{\theta}^2 sin^2(\theta) \tag{4.19}$$

$$= \frac{1}{2}I\dot{\theta}^2 sin^2(\theta) \tag{4.20}$$

and similarly for K_y yields

$$K_y = \frac{1}{2}\int_0^l \frac{m}{l} r^2 dr \dot{\theta}^2 cos^2(\theta) \tag{4.21}$$

$$= \frac{1}{2}I\dot{\theta}^2 cos^2(\theta) \tag{4.22}$$

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

$$K = \frac{1}{2}I\dot{\theta}^2 \tag{4.23}$$

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

$$\begin{aligned} 0 &= \left(\frac{d}{dt}I\dot{\theta}\right) - \left(-mg\frac{l}{2}(\sin(\theta))\right) && (4.27) \\ &= I\ddot{\theta} + mg\frac{l}{2}\sin(\theta). && (4.28) \end{aligned}$$

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	$m_2gl\cos(\theta)$

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

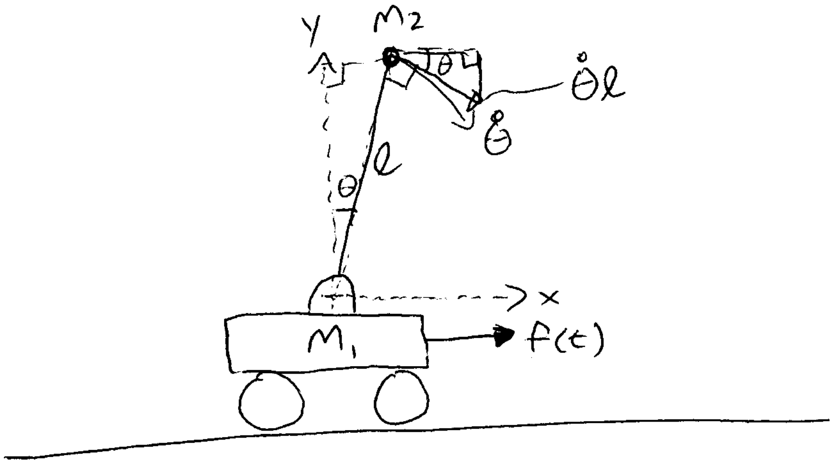


Figure 5.1: Inverted Pendulum Cart.

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.

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

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

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

7.1.1 Derivatives

This is a simple case of integration by parts.

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

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 Convolution

Chapter 8

Z Transform

8.1 One Sided Transform

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

$f(n)$		$F(z)$	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

$$Y(z) = H(z)X(z) \tag{8.5}$$
$$X(z) = \frac{1}{1 - z^{-1}} \tag{8.6}$$
$$H(z) = \frac{1}{1 - \frac{1}{3z}} \tag{8.7}$$

thus

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

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

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

thus

$$\begin{aligned}
\frac{4}{3}z - \frac{1}{3} &= A(z-1) + B\left(z - \frac{1}{3}\right) && (8.17)\\
B &= 1.5 && (8.18)\\
A &= -\frac{1}{6} && (8.19)\\
Y(z) &= 1 + \frac{-\frac{1}{6}}{z-\frac{1}{3}} + \frac{1.5}{z-1} && (8.20)\\
Y(z) &= 1 + \frac{-\frac{1}{6}zz^{-1}}{z-\frac{1}{3}} + \frac{1.5zz^{-1}}{z-1} && (8.21)\\
Y(z) &= 1 - \frac{1}{6}z^{-1}\frac{z}{z-\frac{1}{3}} + 1.5z^{-1}\frac{z}{z-1} && (8.22)\\
y(n) &= \delta(n) + 1.5u(n-1) - \frac{\left(\frac{1}{3}\right)^{n-1}u(n-1)}{6} && (8.23)
\end{aligned}$$

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);
```

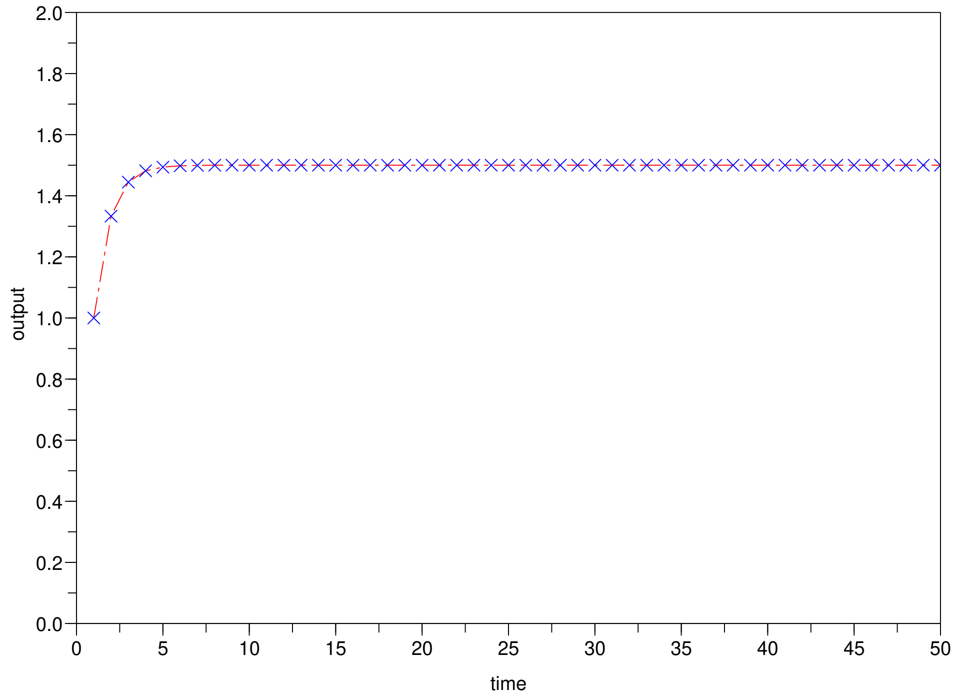
```
// Set up the size of the z-transform to be the same as y1
y2=zeros(2*n-1,1);

// 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

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

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].

$$\hat{x} \quad = \quad \text{argmin}_{\,x} \|Ax - b\| \tag{10.2}$$

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

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

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

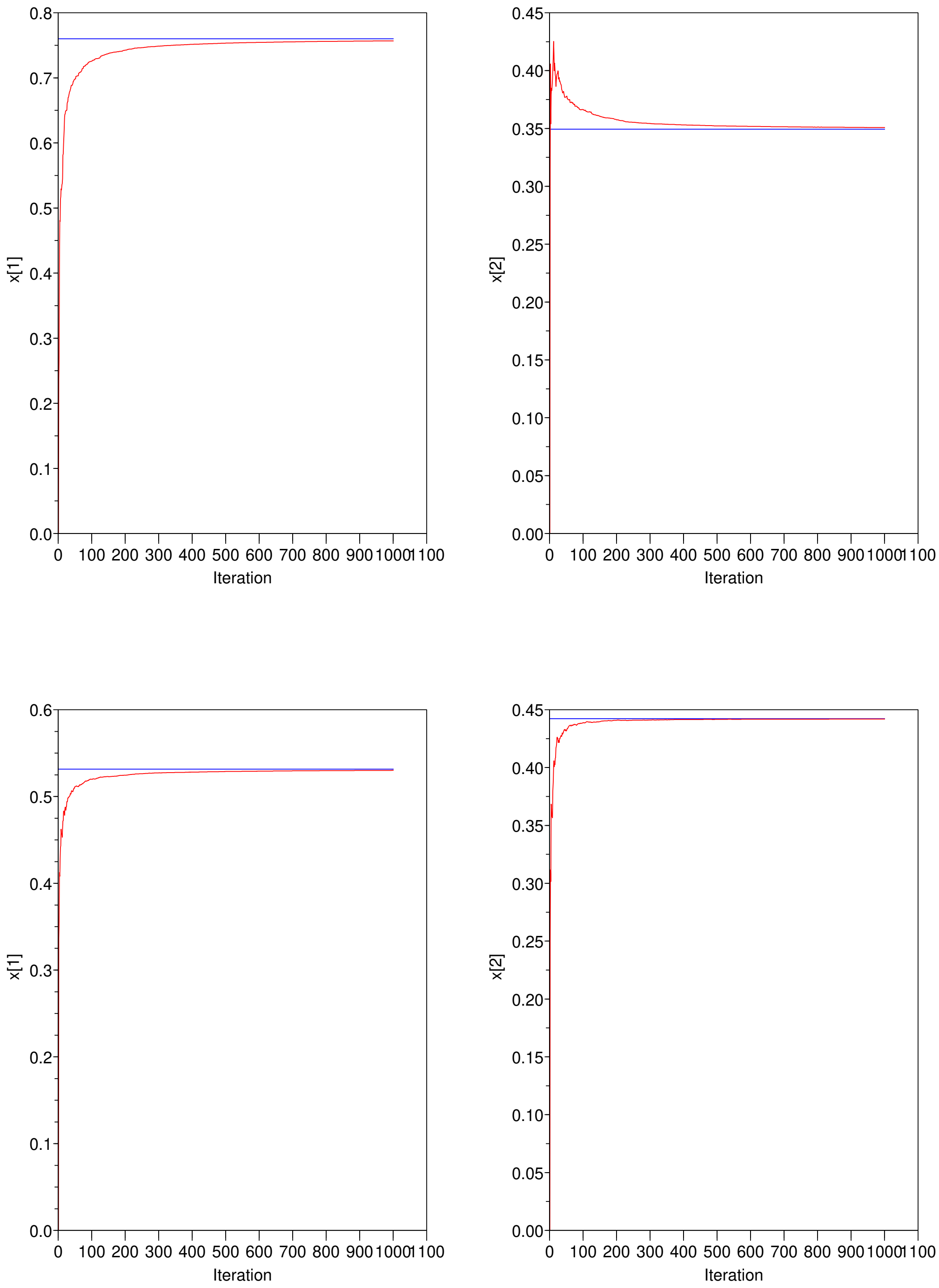
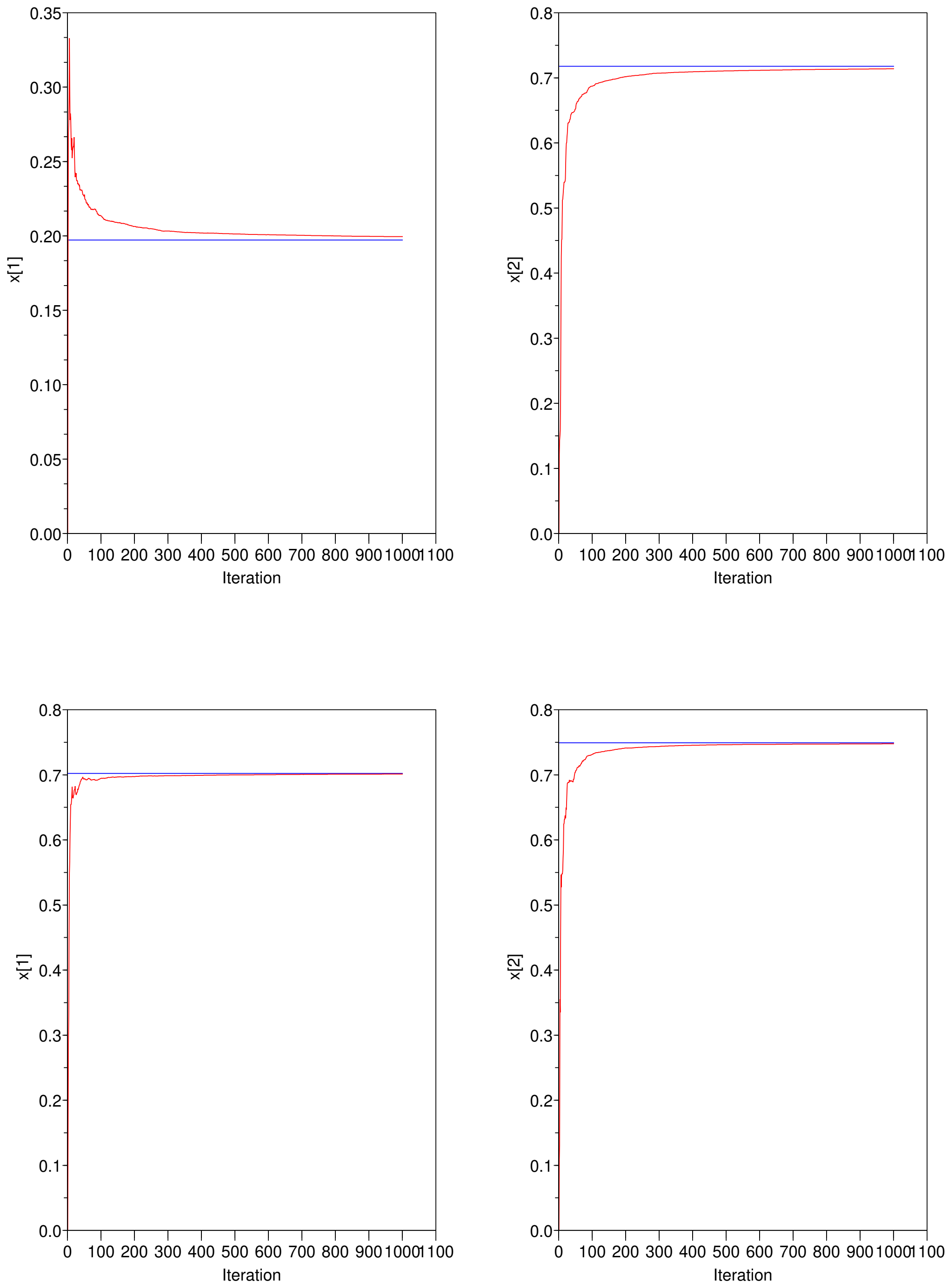


Figure 10.2: More comparisons of “True” parameter values versus estimates for several randomly generated problems



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

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

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]")
```

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

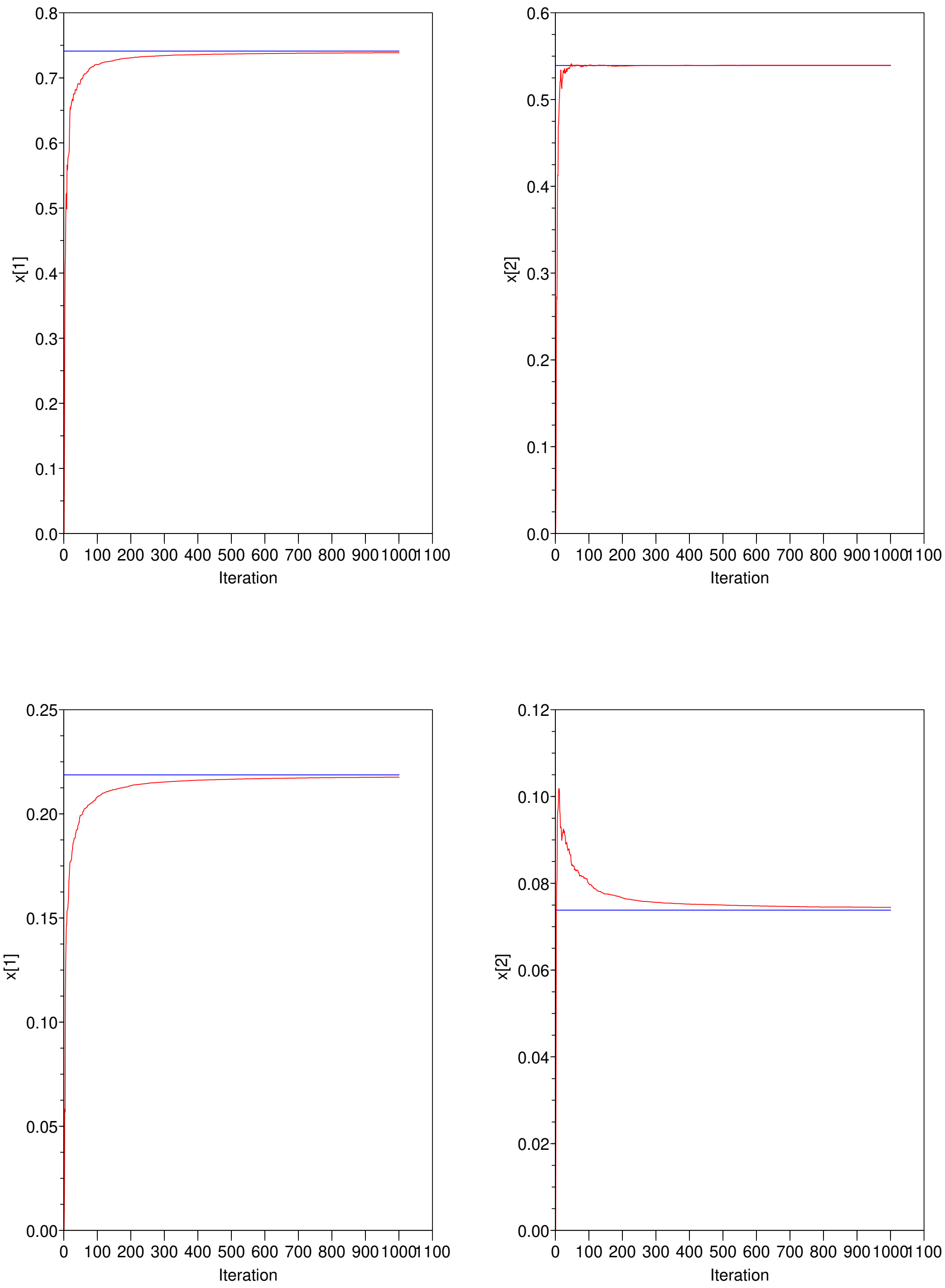
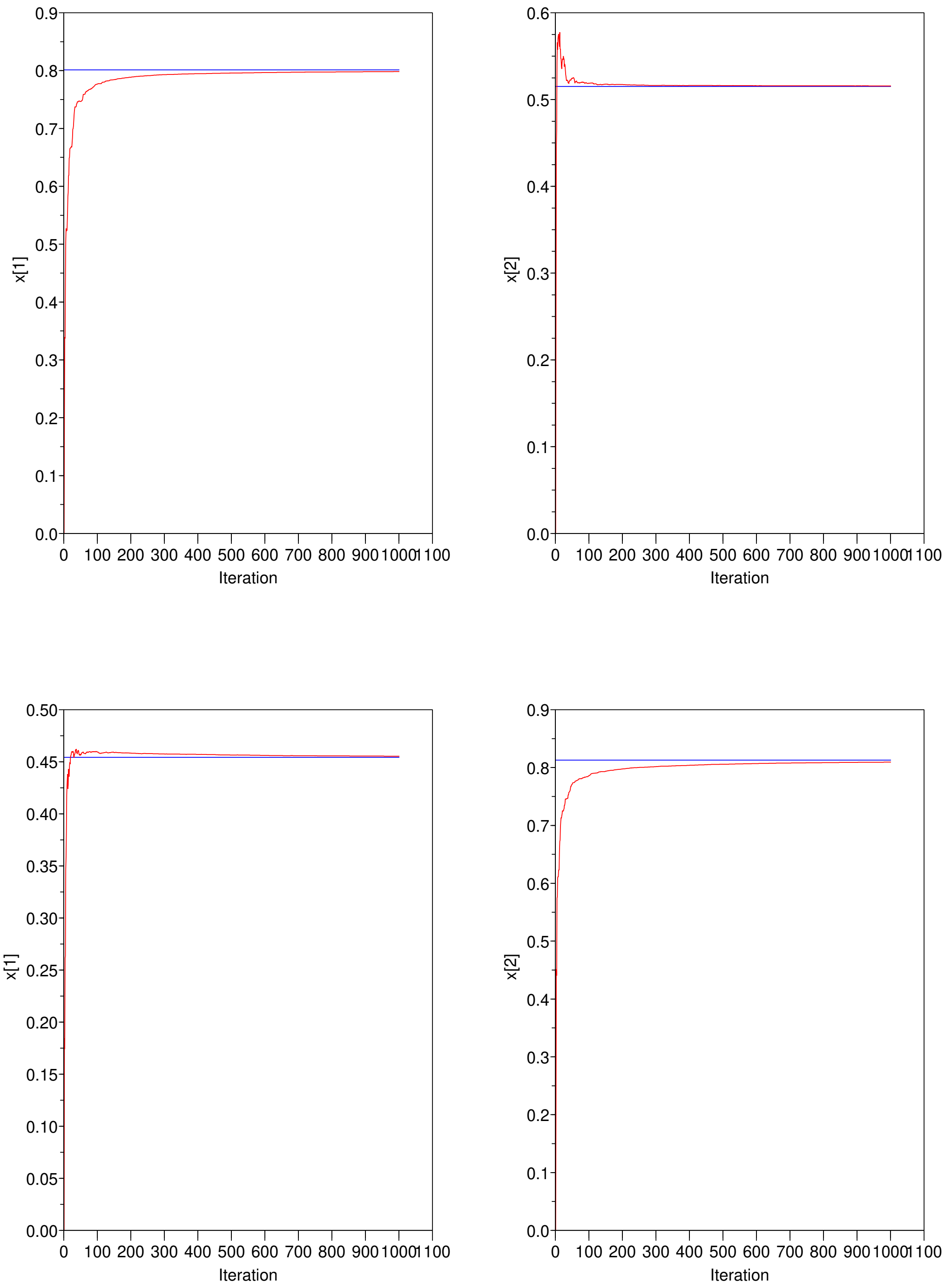


Figure 10.4: More comparisons of "True" parameter values versus estimates for several randomly generated problems



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

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

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

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

The random variables are given by

$$p(w) \sim N(0, R_w) \quad (11.11)$$
$$p(v) \sim N(0, R_v) \quad (11.12)$$

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.

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

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

$$P_{k|k-1} = E[e_{k|k-1} e^T_{k|k-1}] \quad (11.15)$$
$$P_{k|k} = E[e_{k|k} e^T_{k|k}] \quad (11.16)$$

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

$$x_{k+1} = A_k x_k + w_k. \quad (11.17)$$

[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

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

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).

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

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

Now take the derivative with respect to K_k.

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

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

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

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.

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

11.4.1 Correction

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

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

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

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

Similarly we can write the expression for the observations

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

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

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

[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

Imaging Basics

12.1 Convolution Masks

12.2 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].

12.2.1 Differentiation

$$derivative = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \tag{12.1}$$

Sobel

$$S_0 = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \tag{12.2}$$

$$S_1 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{12.3}$$

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

12.2.2 High Pass Filters

$$Laplacian1 = \begin{bmatrix} -1 & 0 & -1 \\ 0 & 4 & 0 \\ -1 & 0 & -1 \end{bmatrix} \tag{12.4}$$

$$Laplacian2 = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \tag{12.5}$$

$$sharpen = \begin{bmatrix} 0 & -1 & 0 \\ -1 & a & -1 \\ 0 & -1 & 0 \end{bmatrix} \tag{12.6}$$

$$a \in \{5, 6, 7\} \tag{12.7}$$

12.3 Histograms

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 12.1: Histogram for Greyscale Image with Arbitrary Shapes.

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

Listing 12.2: 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
```

12.3.1 Finding Regions of Interest

Regions which catch our eyes tend to have different histogram distributions than overall one for the entire image. Consider code 12.3.

Listing 12.3: 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)

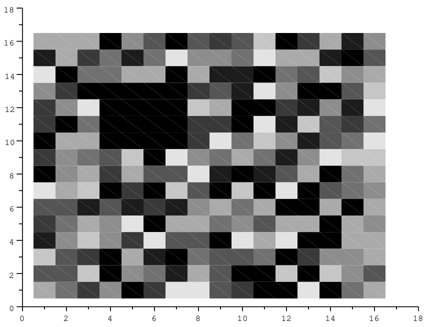
```
A=rand(20,20);
A(4:7,4:7)=ones(4,4);
```

The specific commands to generate each picture were

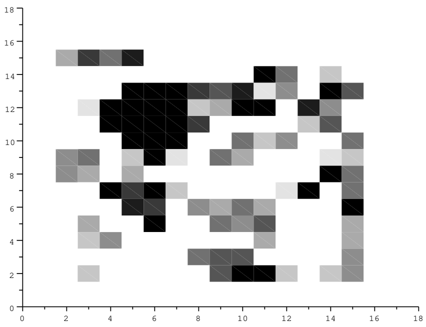
```
f=scf();
Matplot(10-A.*hist_mask(A,threshold=0.3).*10);
f.colormap=graycolormap(10);
```

Table 12.1: Histogram thresholds to yield regions of interest.

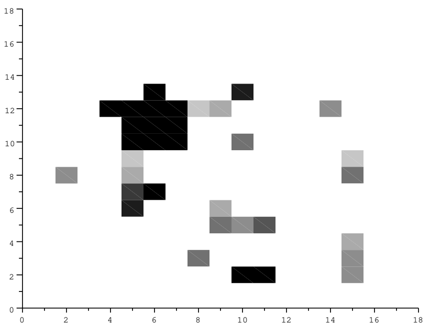
Original



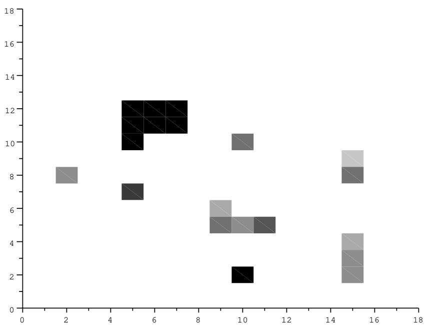
Threshold =0.1



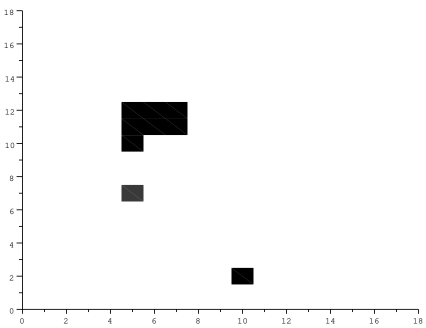
Threshold = 0.3



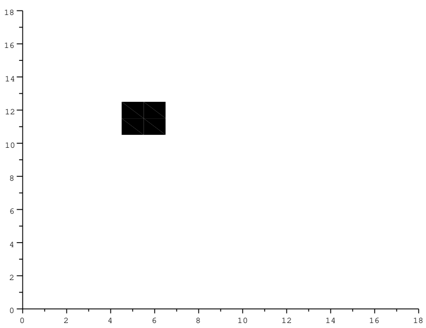
Threshold =0.4



Threshold = 0.5



Threshold =0.6



Part V

Appendices

Appendix A

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.

A.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

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

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

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

A.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.

A.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

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

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

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

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

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

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

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

The second element is

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

The third element is

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

The fourth element is

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

Then repeat the process till every box is filled.

A.3 Matrix Classification

A.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	$A_{ij} = 0 \forall \lvert i - j \rvert \leqslant 1$
Pentediagonal	$A_{ij} = 0 \forall \lvert i - j \rvert \leqslant 3$
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 ($A > 0$)	$x^H Ax > 0 \forall x \in \mathbb{C}^n$

A.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

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

[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 77.

for $V = U^H Q$. Note that $V^H \Sigma V$ must be symmetric positive definite, thus $x^H V^H \Sigma V x \geqslant \|x\|_2^2 \sigma_n$. 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

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

Now factor this expression as follows

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

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 B

Using SciLab

B.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.

B.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.

B.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