Problem 5.2

i) minimize $\|\underline{x}\|_2^2$ subject to $\underline{a}^T\underline{x} = \underline{b}$

critical point

The problem is convex: both the objective function and constraints are convex. Therefore, the solution is unique and is a global min.

Note: $\|x\|_2^2 = x_1^2 + x_2^2 + \dots + x_n^2$ and sum of convex functions is convex.

$$\Lambda(\underline{x}, \underline{\lambda}) = \|\underline{x}\|_2^2 - \underline{\lambda}^T(\underline{a}^T\underline{a} - \underline{b})$$

λ: scalar

$$\frac{\partial \Lambda}{\partial \underline{x}} = 2\underline{x} - \lambda\underline{a} = 0 \Rightarrow \underline{x} = \frac{\lambda}{2}\underline{a} \quad [1]$$

$$\frac{\partial \Lambda}{\partial \underline{\lambda}} = -(\underline{a}^T\underline{a} - \underline{b}) = 0 \Rightarrow \underline{a}^T\underline{x} = \underline{b} \quad [2]$$

$$\underline{a}^T\left(\frac{\lambda}{2}\underline{a}\right) = \underline{b}$$

$$\lambda * \underline{a}^T\underline{a} = 2\underline{b}$$

$$\lambda = \frac{2\underline{b}}{\underline{a}^T\underline{a}}$$

$$\therefore \underline{x}^* = \frac{\lambda}{2}\underline{a} = \left(\frac{1}{2}\right)\left(\frac{2\underline{b}}{\underline{a}^T\underline{a}}\underline{a}\right) = \frac{\underline{b}}{\underline{a}^T\underline{a}}\underline{a}$$ (critical value)

$$\|\underline{x}^*\|_2^2 = \left\|\frac{\underline{b}}{\underline{a}^T\underline{a}}\underline{a}\right\|_2^2 = \frac{\underline{b}^T\underline{a}^T\underline{a}}{(\underline{a}^T\underline{a})^2} = \frac{\underline{b}^2}{\underline{a}^T\underline{a}}$$

$$\therefore \boxed{\|\underline{x}^*\|_2^2 = \frac{\underline{b}^2}{\underline{a}^T\underline{a}}}$$

Problem 5.2

a) $h_k \ll 1$, $\sin h_k \approx h_k$

$v_k = v + k$.

$$y_{k+1} = y_k + h v_k \sin h_k \approx y_k + h v_k h_k$$
$$h_{k+1} = h_k + h w_k$$

$$\therefore x_{k+1} = \begin{bmatrix} 1 & h v_k \\ 0 & 1 \end{bmatrix} \begin{bmatrix} y_k \\ h_k \end{bmatrix} + \begin{bmatrix} 0 \\ h \end{bmatrix} w_k$$

$$= \underbrace{\begin{bmatrix} 1 & hv \\ 0 & 1 \end{bmatrix}}_{a} \underline{x}_k + \underbrace{\begin{bmatrix} 0 \\ h \end{bmatrix}}_{b} w_k$$

b) $\underline{x} = [w_0, w_1, w_2, \underline{x}_1^T, \underline{x}_2^T, \underline{x}_3^T]^T$

$$f(\underline{x}) = \underline{x}_1^T Q \underline{x}_1 + \underline{x}_2^T Q \underline{x}_2 + \underline{x}_3^T Q \underline{x}_3 + r w_0^2 + r w_1^2 + r w_2^2$$

this can be expressed concisely as:

$$f(\underline{x}) = [w_0, w_1, w_2, \underline{x}_1^T, \underline{x}_2^T, \underline{x}_3^T] \underbrace{\begin{bmatrix} r & & & & & \\ & r & & & & \\ & & \underline{Q} & & & \\ & & & \underline{Q} & & \\ & & & & \underline{Q} & \\ & & & & & \underline{Q} \end{bmatrix}}_{M} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \underline{x}_1^T \\ \underline{x}_2^T \\ \underline{x}_3^T \end{bmatrix}$$

$$= \underline{x}^T \begin{bmatrix} r & & & & & \\ & r & & & & \\ & & \underline{Q} & & & \\ & & & \underline{Q} & & \\ & & & & \underline{Q} & \\ & & & & & \underline{Q} \end{bmatrix} \underline{x}$$

where $\underline{Q}$ are block matrices.

$$\underline{x}_1 = \underline{A}\underline{x}_0 + \underline{B}w_0$$

$$= \begin{bmatrix} 1 & hv_1 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} y_0 \\ h_0 \end{bmatrix} + \begin{bmatrix} 0 \\ h \end{bmatrix} w_0$$

$$\Rightarrow \begin{aligned} y_1 &= y_0 + hv h_0 \\ h_1 &= h_0 + h w_0 \end{aligned}$$

$$\underline{x}_2 = \underline{A}\underline{x}_1 + \underline{B}w_1$$

$$\Rightarrow \begin{aligned} y_2 &= y_1 + hv h_1 \\ h_2 &= h_1 + h w_1 \end{aligned}$$

$$\underline{x}_3 = \underline{A}\underline{x}_2 + \underline{B}w_2$$

$$\Rightarrow \begin{aligned} y_3 &= y_2 + hv h_2 \\ h_3 &= h_2 + h w_2 \end{aligned}$$

$(y_1 h_1)$ $(y_2 h_2)$ $(y_3 h_3)$

$$\Rightarrow \begin{aligned} y_1 - y_0 - hv h_0 &= 0 \\ h_1 - h_0 - h w_0 &= 0 \\ y_2 - y_1 - hv h_1 &= 0 \\ h_2 - h_1 - h w_1 &= 0 \\ y_3 - y_2 - hv h_2 &= 0 \\ h_3 - h_2 - h w_2 &= 0 \end{aligned}$$

$w_0\ w_1\ w_2\ x_1\ x_2\ x_3$

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ -h_0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & -hv & 1 & 0 & 0 & 0 \\ 0 & -h & 0 & 0 & -1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & -hv & 1 & 0 \\ 0 & 0 & -h & 0 & 0 & 0 & -1 & 0 & 1 \end{bmatrix}$$

$$\therefore g(\underline{x}) = \underline{R}\,\underline{x} + \begin{bmatrix} -y_0 - hv h_0 \\ -h_0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \underline{0}$$

$\underline{R}$

$\underline{r}$

6×9 $\quad 9 \times 1$

6×1 $\qquad 6 \times 1$

$$h(\underline{x}) = \begin{bmatrix} w_k - w_{min} \\ -w_k + w_{max} \end{bmatrix} \Bigg|_{k=0,1,2} \geq 0$$

We have:
$$\begin{aligned} w_0 - w_{min} &\geq 0 \\ -w_0 + w_{max} &\geq 0 \\ w_1 - w_{min} &\geq 0 \\ -w_1 + w_{max} &\geq 0 \\ w_2 - w_{min} &\geq 0 \\ -w_2 + w_{max} &\geq 0 \end{aligned}$$

in the form $h(\underline{x}) = \underline{S}\underline{x} + \underline{s} \geq 0$

$$h(\underline{x}) = \begin{bmatrix} w_0 & w_1 & w_2 & x_1 & x_2 & x_3 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 \end{bmatrix} \underline{x} + \begin{bmatrix} -w_{min} \\ w_{max} \\ -w_{min} \\ w_{max} \\ -w_{min} \\ +w_{max} \end{bmatrix} \geq \underline{0}$$

$\underline{S}$ $\quad \underline{0} = [0\ 0]$ $\quad \underline{s}$

We may now state:

$$\min_{\underline{x}} f(\underline{x}) = \underline{x}^T \underline{M} \underline{x} \text{ subject to } g(\underline{x}) = \underline{R}\underline{x} + \underline{c} = \underline{0}$$

and $h(\underline{x}) = \underline{S}\underline{x} + \underline{s} \geq 0$

c) A problem is convex if the objective function is convex and the feasible set is convex.

→ Check if objective function is convex

$$f(x) = \underline{x}_1^T Q \underline{x}_1 + \underline{x}_2^T Q \underline{x}_2 + \underline{x}_3^T Q \underline{x}_3 + r w_0^2 + r w_1^2 + r w_2^2$$

Note that $\underline{x}_k^T Q \underline{x}_k$ reduces to a sum of quadratic terms.

i.e $\underline{x}_k^T Q \underline{x}_k = \begin{bmatrix} y_k & h_k \end{bmatrix} \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} y_k \\ h_k \end{bmatrix}$

$$= \begin{bmatrix} a y_k + c y_k & b h_k + d h_k \end{bmatrix} \begin{bmatrix} y_k \\ h_k \end{bmatrix}$$

$$= a y_k^2 + c y_k^2 + b h_k^2 + d h_k^2.$$

Also, $r w_k^2$ are also quadratic terms.

Claim: Any quadratic function is convex.

Proof: Assume $y = ax^2 + b = f(x)$

$$f\left((1-\alpha)x_1 + \alpha x_2\right) = a\left((1-\alpha)x_1 + \alpha x_2\right)^2 + b$$

$$= a\left[(1-\alpha)^2 x_1^2 + 2(1-\alpha)(\alpha) x_1 x_2 + \alpha^2 x_2^2\right] + b.$$

$$\leq a\left[(1-\alpha)^2 x_1^2 + \alpha^2 x_2^2\right] + b$$

$$\leq a\left[(1-\alpha) x_1^2 + \alpha x_2^2\right] + b$$

Note $\left[b = (1-\alpha)b + \alpha b\right]$

$$\leq a(1-\alpha) x_1^2 + (1-\alpha) b + \alpha \cdot a x_2^2 + \alpha b.$$

$$= (1-\alpha)\left(a x_1^2 + b\right) + \alpha\left(a x_2^2 + b\right)$$

$$= (1-\alpha) f(x_1) + \alpha f(x_2)$$

as required.

Since the sum of quadratic (or in general convex) functions is also convex, as we have proved last assignment, the objective function is also convex. ∎

→ the equality and inequality constraints define a convex set as:

equality: our equality constraint is of the form $\underline{A}\underline{x} + \underline{b} = 0$.

$$\Rightarrow \underline{A}\underline{x} = -\underline{b}$$

A set $S \subseteq \mathbb{R}^n$ is convex if for any $\underline{x}, \underline{y} \in S$, the point $t\underline{x} + (1-t)\underline{y}$ is also in S for any $t \in [0,1]$.

$$\Rightarrow A(t\underline{x} + (1-t)\underline{y}) = t\underline{A}\underline{x} + (1-t)\underline{A}\underline{y}$$
$$= t(-\underline{b}) + (1-t)(-\underline{b})$$
$$= -\underline{b}t + -\underline{b} + t\underline{b}$$
$$= -\underline{b} \text{ since } \underline{A}\underline{x} = -\underline{b}, \underline{A}\underline{y} = -\underline{b}.$$

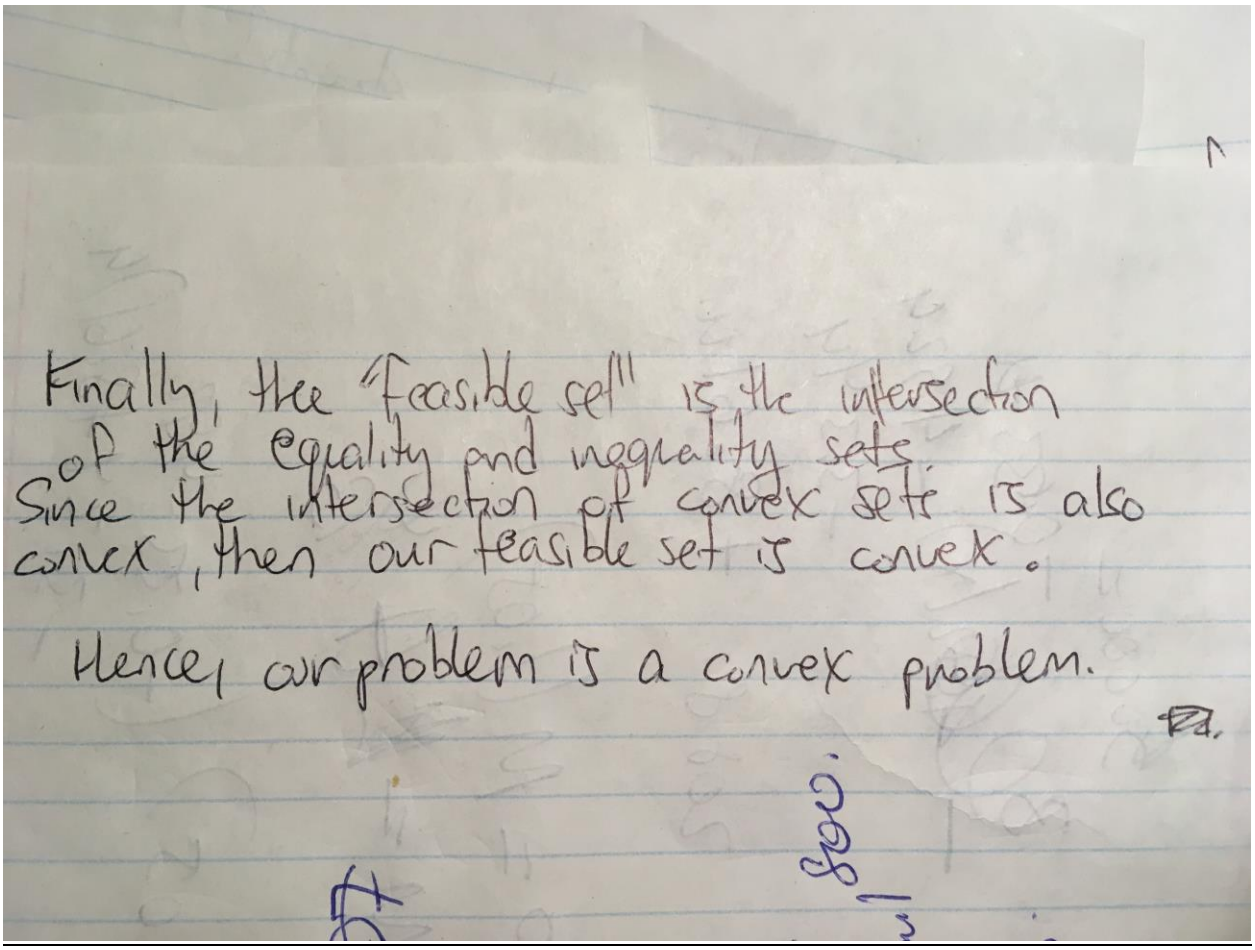
∴ since $A(t\underline{x} + (1-t)\underline{y}) = -\underline{b}$, then $t\underline{x} + (1-t)\underline{y}$ must also be in S.
∴ equality constraint defines a convex set.

Similarly, our inequality constraint can be brought into the form $\underline{A}\underline{x} + \underline{b} \geq 0$.

$$\underline{A}\underline{x} \geq -\underline{b}$$

$$A(t\underline{x} + (1-t)\underline{y}) = t\underline{A}\underline{x} + (1-t)\underline{A}\underline{y}$$
$$\geq t(-\underline{b}) + (1-t)(-\underline{b})$$
$$\geq -\underline{b}.$$

∴ inequality constraint also defines a convex set.



Problem 5.2

d)

We use the sparse matrix to save memory space and speed up computation time. Without it, MatLab takes long to run. In order to implement Matlab's quadprog function, we need to bring the form into `1/2*x'*H*x + f'*x`, where H contains all the quadratic terms and f contains all the linear terms. We do not have any linear terms in our problem, so that will simply be a zero matrix. As can be seen in c), the problem has already been restructured into the form $x^T M x$, so all that's left is to implement the M matrix and to double it. M is simply a diagonal matrix with block matrices along its diagonal, which can be viewed as an array of column vectors with the first N columns with r along the diagonal, and N columns of the block matrix Q along the diagonal. Since each block Q is actually 2 by 2, then it is essentially 2*N columns. Therefore, our H matrix is of size N + 2*N = 3*N = 3000. This can be implemented recursively, using the Matlab function blkdiag which essentially creates a diagonal matrix with its parameters as the diagonal block matrices. For example, in the first iteration of the for-loop, H is a diagonal matrix with R and Q along the diagonal. R is a block matrix of size 1000 with r along the diagonal, and Q is a 2 by 2 matrix. In the second iteration, H is diagonal matrix with the previous H *and* another Q matrix along the diagonal – in other words, the original R matrix with two Q block diagonals. Iterating this for N=1000, we obtain the desired H matrix with the diagonal as the original R matrix and 1000 Q block matrices.

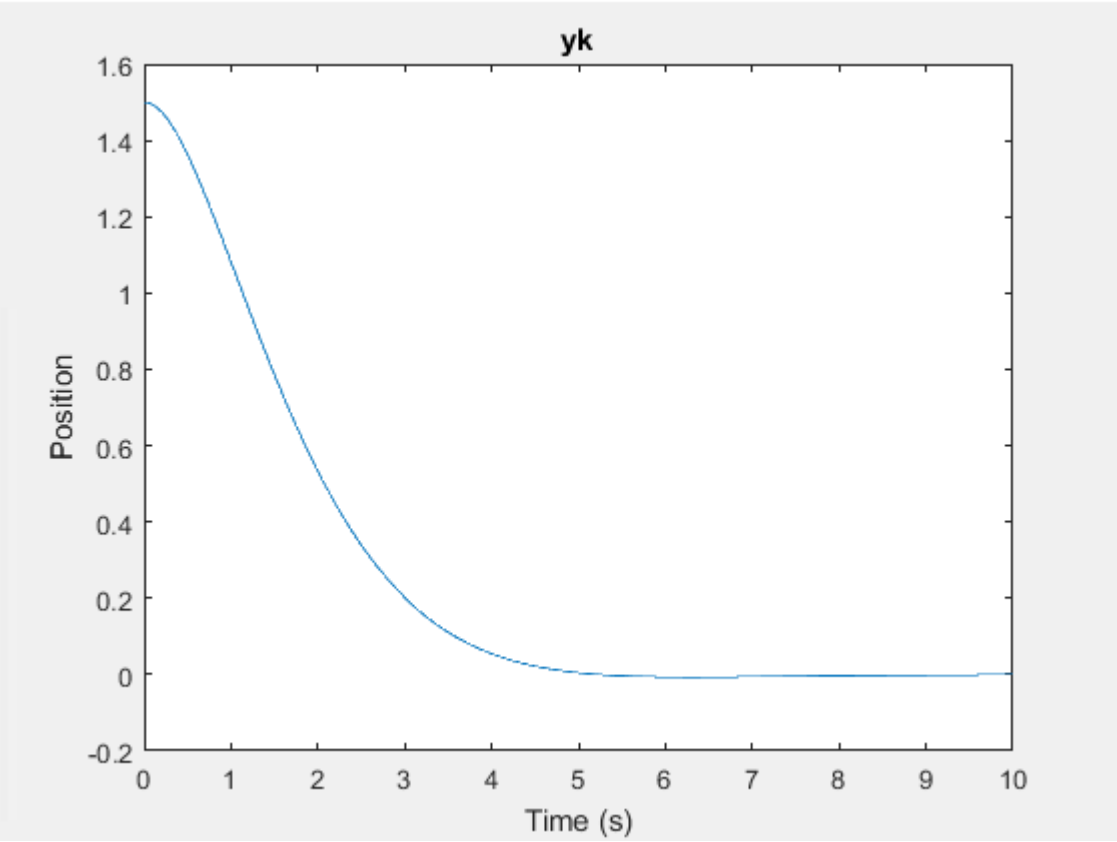
The inequality constraint is brought into the form Ax<B. It is slightly restructured from c) in that the order in which the equations $w_k - w_{min} \geq 0$, etc. are placed in the matrix need not matter. In this view, the B vector is simply all the $-w_{min}$ at the top, and all the w_{max} at the bottom. There are N equations for w_{min} and N equations for w_{max}, therefore B and x are of size 2N. Furthermore, A is a 2000 by 3000 matrix, and can be represented by four (inequal) block matrices: $\begin{matrix} A & B \\ C & D \end{matrix}$, where, as mentioned before, A is an identity matrix of size N by N, C is a negative identity matrix of size N by N, B is a 0 matrix of size N by 2N, D is a 0 matrix of size N by 2N.

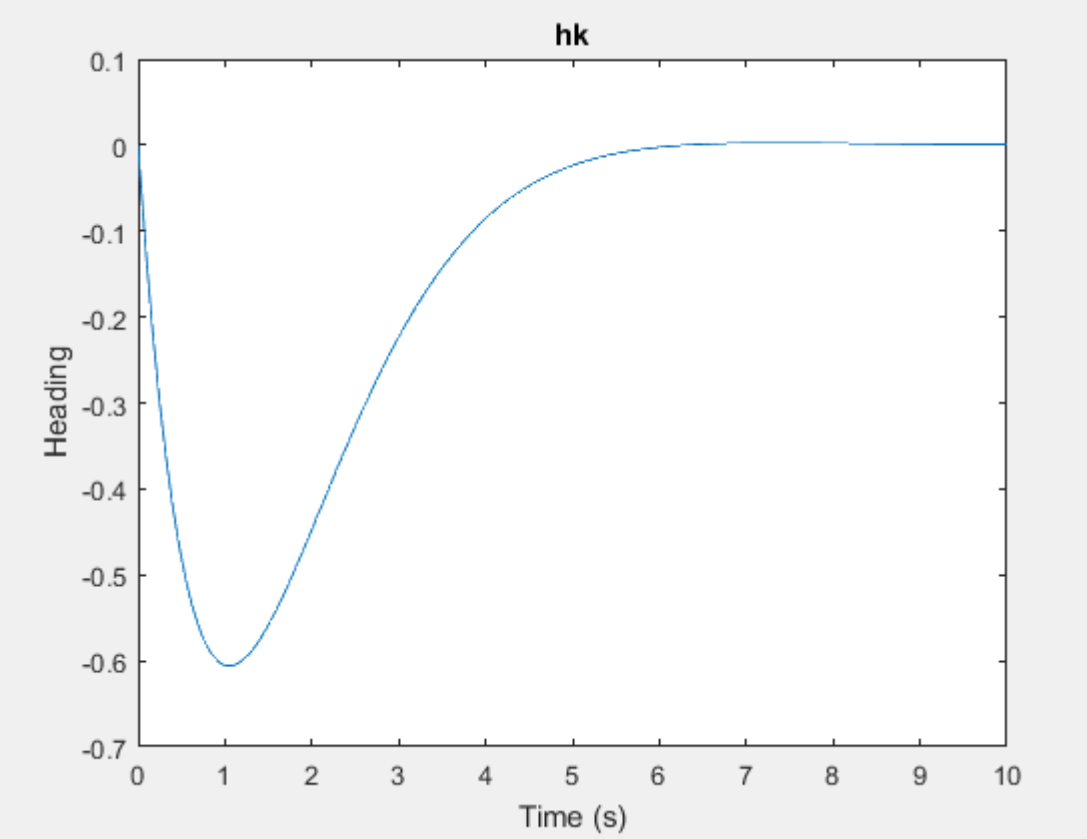
For brevity, the justification for the equality constraint is omitted.

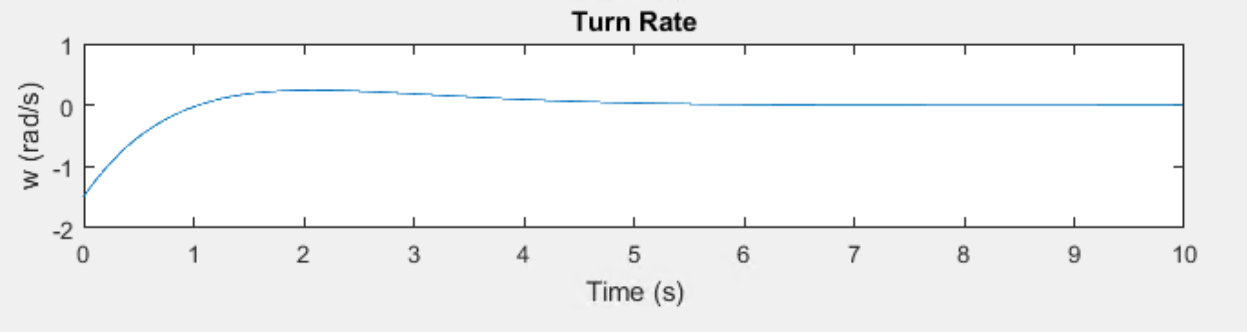
```
clear all;
clc;

N=1000;
h=0.01;
v=1;
wmin=-pi*4;
wmax=pi*4;

%initial conditions
y0=1.5;
h0=0;
r=1;
Q=[1 0; 0 1];

%x = quadprog(H,f,A,b,Aeq,beq,lb,ub,x0)
%objective function
H = sparse(eye(N)*r); %identity matrix with size N with diagonal r
for i = 1:N
 H = blkdiag(H, Q);
end
H = 2 * H;
f = zeros(3*N,1);

%inequality constraint
%Aineq: Matrix in linear inequality constraints Aineq*x ? bineq
%bineq: Vector in linear inequality constraints Aineq*x ? bineq
Aineq = sparse([eye(N) zeros(N,2*N);-1*eye(N) zeros(N,2*N)]);
bineq = sparse([ones(N,1)*(-wmin); ones(N,1)*wmax]);

%equality constraint
%Aeq: Matrix in linear equality constraints Aeq*x = beq
%beq: Vector in linear equality constraints Aeq*x = beq


LEFTMATRIX = zeros(2*N,N);
for i = 1:N
    LEFTMATRIX(2*i,i)=-h;
end

RIGHTMATRIX = zeros(2*N,2*N);
```

```
for i = 1:2*N
    if mod(i,2) == 1 %if odd
        RIGHTMATRIX(i,i)=1;
            if (i<2*N-1)%to prevent out of indexing
            RIGHTMATRIX(i+2,i)=-1;
            end
    else %if even
        if i<2*N-1
            RIGHTMATRIX(i,i)=1;
            RIGHTMATRIX(i+1,i)=-h*v;
            RIGHTMATRIX(i+2,i)=-1;
        elseif i<2*N
            RIGHTMATRIX(i,i)=1;
            RIGHTMATRIX(i+1,i)=-h*v;
        else %i=2*N
            RIGHTMATRIX(i,i)=1;
        end
    end
end

Aeq = [LEFTMATRIX RIGHTMATRIX];
beq = [y0+h*v*h0; h0; zeros(2*N-2,1)];

[x, fval] = quadprog(H,f,Aineq,bineq,Aeq,beq);
% x = [w0 w1 ... wn x1 x2 ...xn]

w = x(1:N);

yk = zeros(1,1000); %initializing
hk = zeros(1,1000);

%construct {yk}
loopcounter=1;
for i=N+1:3*N
    if mod(i,2)==1
        yk(loopcounter)= x(i);
        loopcounter = loopcounter +1;
    end
end

%construct {hk}

loopcounter1=1;
for i=N+1:3*N
    if mod(i,2)==0
        hk(loopcounter1)= x(i);
        loopcounter1 = loopcounter1 +1;
    end
end
t=(0:N)*h;
subplot(2,1,1);
plot(t,[y0 yk]);
title('{yk}');
xlabel('Time (s)')
ylabel('Position');
```

```
subplot(2,1,2);
plot(t, [h0, hk]);
title('{hk}');
xlabel('Time (s)');
ylabel('Heading');
```




hk
Heading
Time (s)
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-0.6
-0.7
0
1
2
3
4
5
6
7
8
9
10

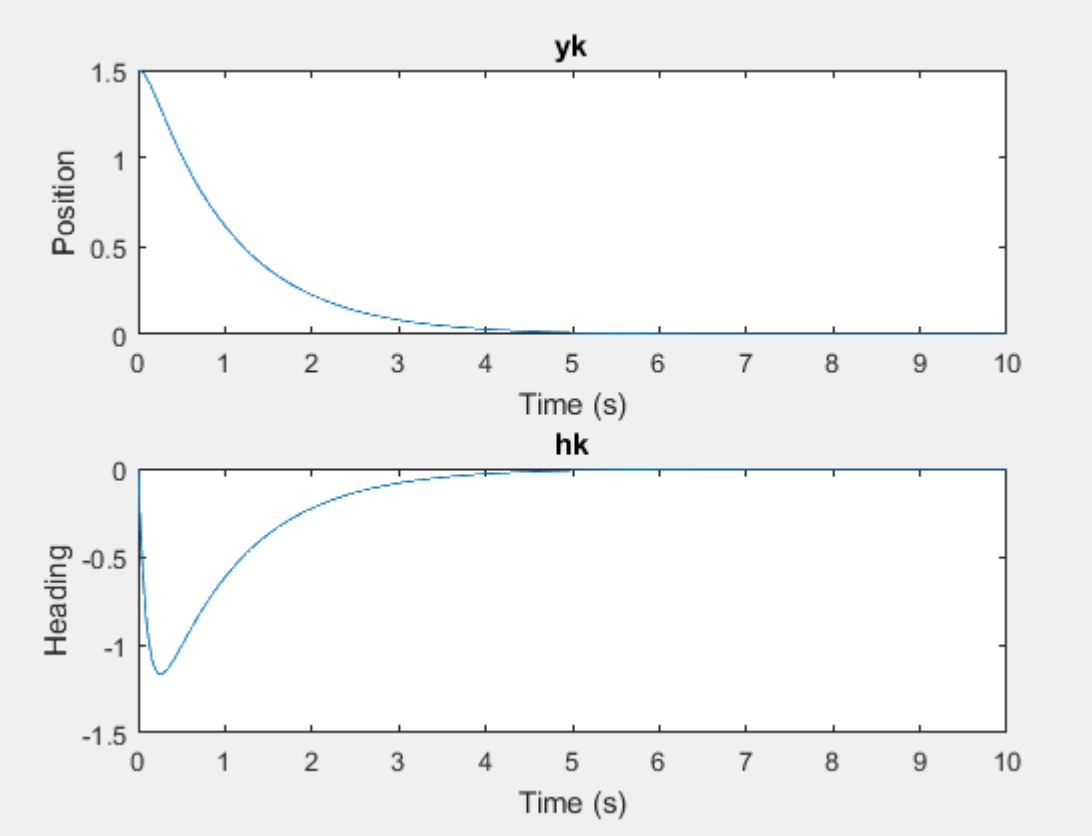

Turn Rate
w (rad/s)
Time (s)
1
0
-1
-2
0
1
2
3
4
5
6
7
8
9
10

e)

Q will be changed to the following:

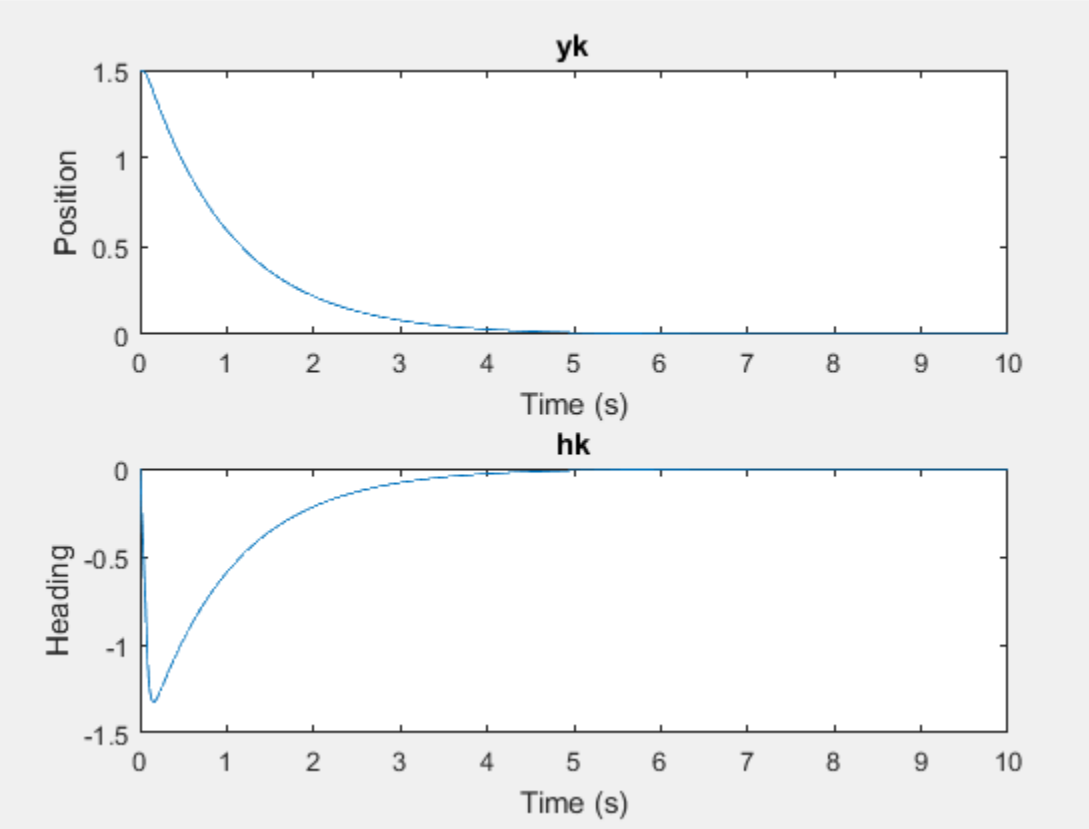
i) Constant diagonals:

$$\begin{bmatrix} 100 & 0 \\ 0 & 100 \end{bmatrix}$$



Expanding out the cost function, it can be seen that the element in the first row and column corresponds to the position (weights it more) and the last row and column element corresponds to the heading. If we weight them equally, both at 100, but scaled 100x more than the initial, we see that the position remains more or less the same, but the heading definitely is more concave (sharper bottom peak).

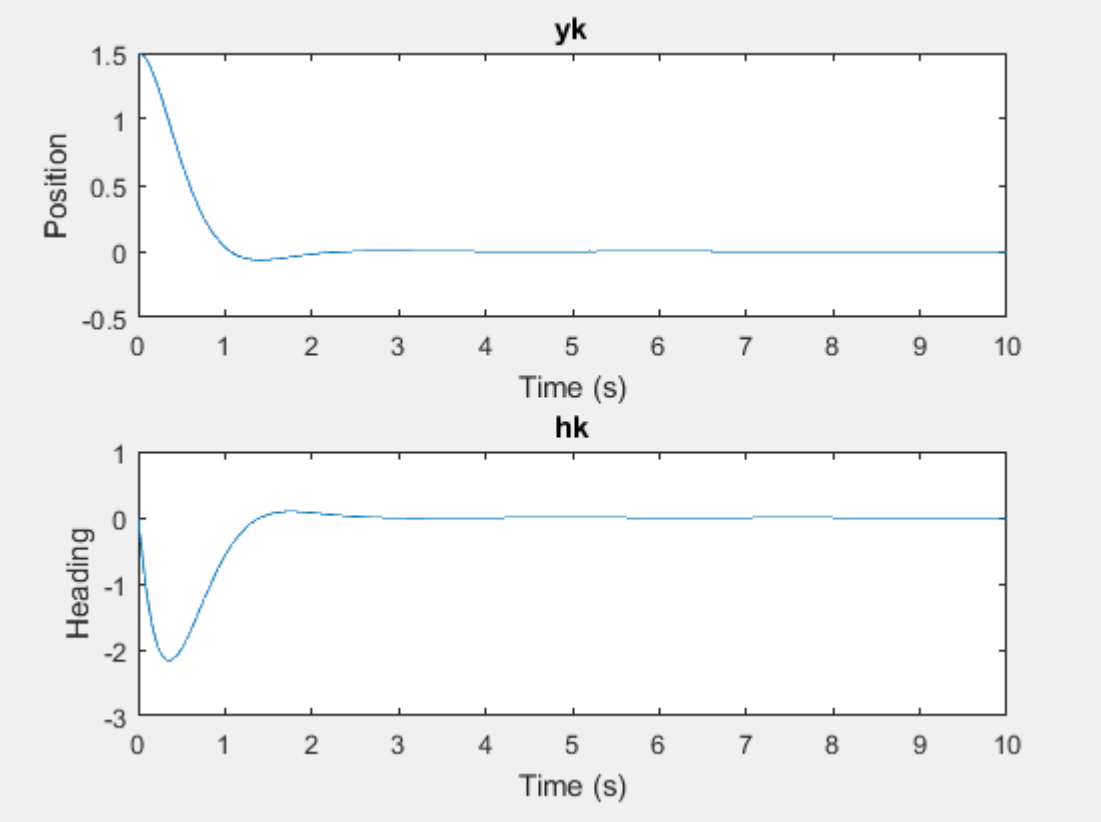
$$\begin{bmatrix} 1000 & 0 \\ 0 & 1000 \end{bmatrix}$$



Weighing both factors even more, we can see that the position converges one second faster than the first case, and the heading has an even spikier bottom peak. Since the goal is to move the robot along a straight line, by optimizing, we are finding the best inputs that would allow the robot to move as close to the line as possible. Therefore, we can view both position and heading as the error or deviation from the desired trajectory.

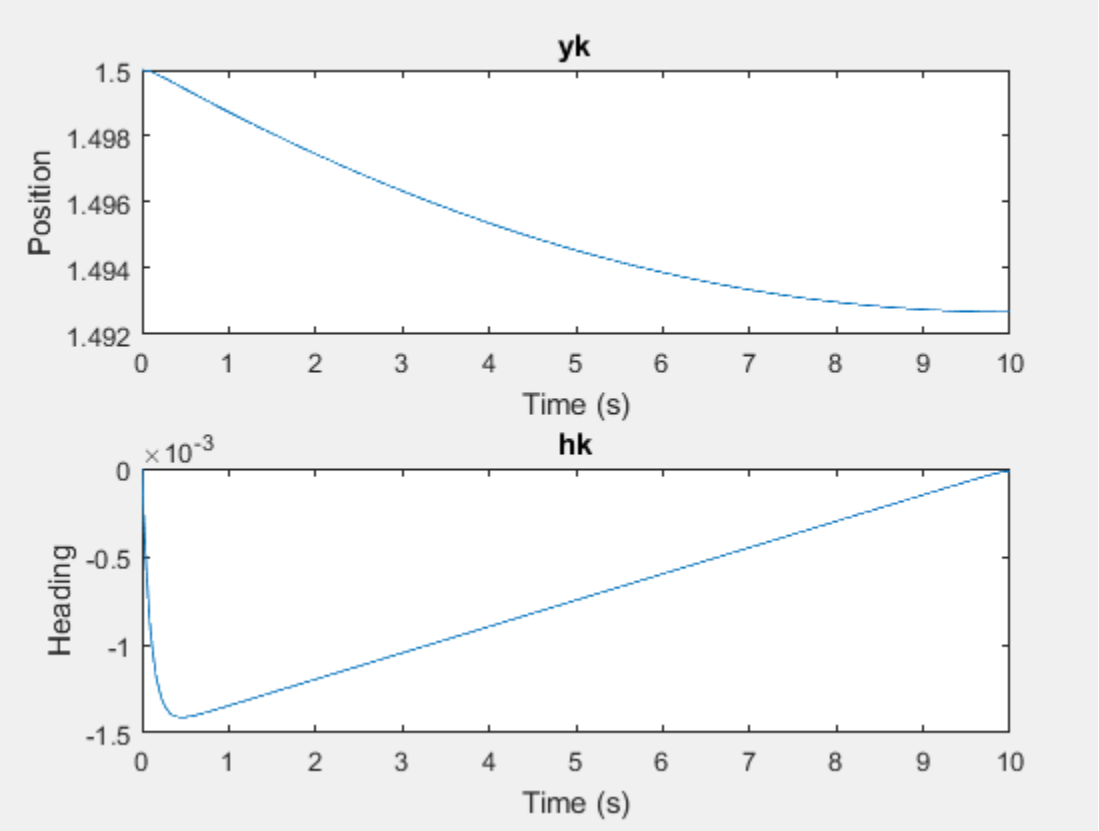
ii) Varying diagonals:

$$\begin{bmatrix} 100 & 0 \\ 0 & 0.01 \end{bmatrix}$$



Scaling the position 1000x more than the heading, we see that position converges very quickly – in 1 second. The heading also converges to 0 very quickly too, but there's some sort of minutiae oscillation about 0 after convergence, which may be a result of an imbalance between the heading and position weightings.

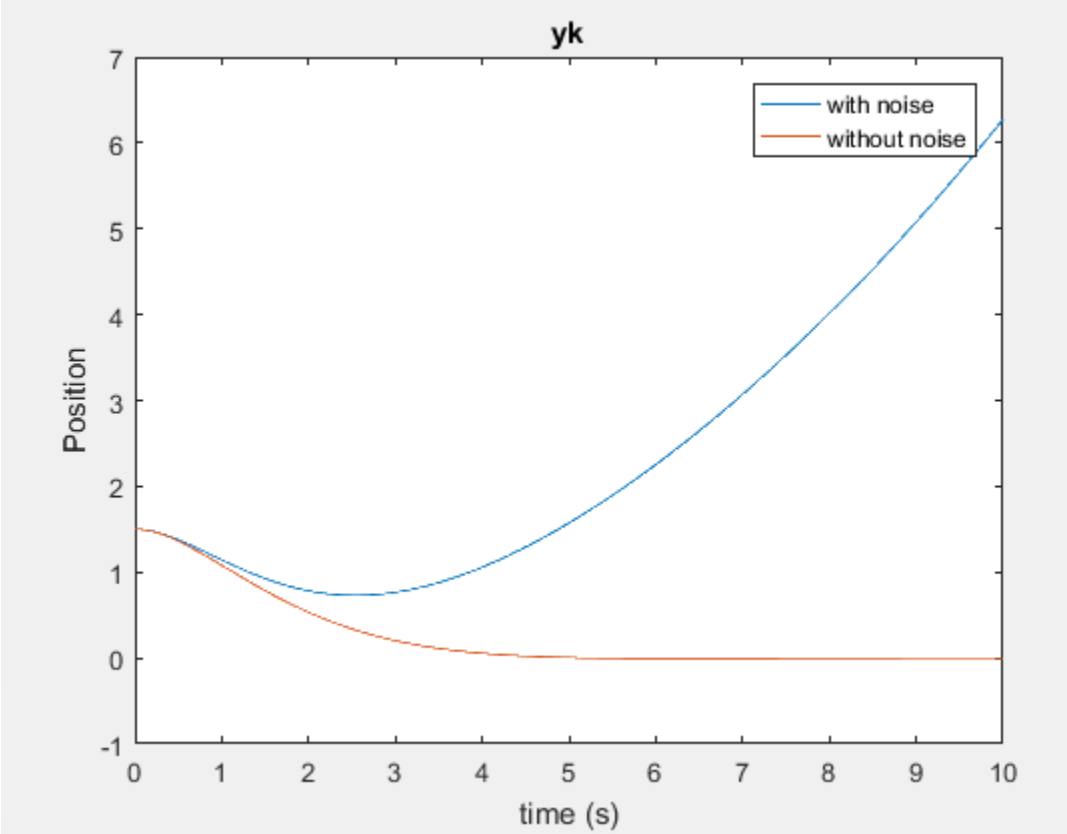
$$\begin{bmatrix} 0.01 & 0 \\ 0 & 100 \end{bmatrix}$$

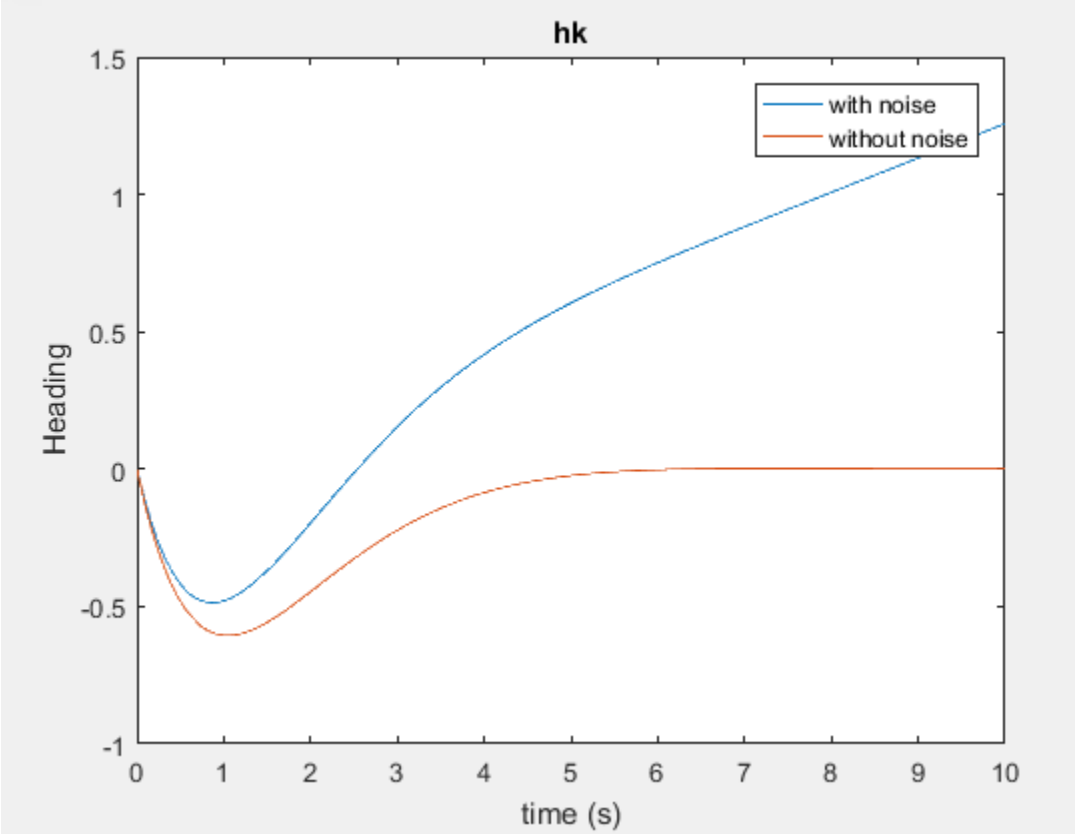


Now, the opposite scenario, having the heading quadratic term weighted much more than the position quadratic term. Convergence is extremely slow and it does not reach to 0. The heading error is almost 0 – the scale is in the order of 10^-3.

f)

```
%simulation with noise


y=zeros(1,N+1);
head=zeros(1,N+1);
y(1)=1.5;
head(1)=0;

for i=1:N
   y(i+1)=y(i)+h*v*head(i);
   head(i+1)=head(i)+h*w(i)+h*pi/25;
end

t=(0:N)*h;
plot(t,y);
hold on;
plot(t,[y0 yk]);
legend('with noise', 'without noise');
xlabel('time (s)');
ylabel('Position');
title('{yk}');

plot(t,head);
hold on;
plot(t,[h0 hk]);
legend('with noise', 'without noise');
xlabel('time (s)');
ylabel('Heading');
title('{hk}');
```


yk
with noise
without noise
Position
time (s)
7
6
5
4
3
2
1
0
-1
0
1
2
3
4
5
6
7
8
9
10


hk
with noise
without noise
Heading
time (s)
1.5
1
0.5
0
-0.5
-1
0
1
2
3
4
5
6
7
8
9
10

The two graphs are compared on the same axis. The difference between this scenario from the last is that there is an extra disturbance or noise factor. This error accumulates over time and hence, diverges – the robot never reaches its desired trajectory of a straight line.

g)

```
%%
%BONUS
%optimization in each step

clear all;
clc;

N=1000;
h=0.01;
v=1;
wmin=-pi*4;
wmax=pi*4;

%initial conditions
y0=1.5;
h0=0;
r=1;
Q=[1 0; 0 1];
%%
%x = quadprog(H,f,A,b,Aeq,beq,lb,ub,x0)
%objective function
H = sparse(eye(N)*r); %identity matrix with size N with diagonal r
for i = 1:N
 H = blkdiag(H, Q);
end
H = 2 * H;
f = zeros(3*N,1);
%%
%inequality constraint
%Aineq: Matrix in linear inequality constraints Aineq*x ? bineq
%bineq: Vector in linear inequality constraints Aineq*x ? bineq
Aineq = sparse([eye(N) zeros(N,2*N);-1*eye(N) zeros(N,2*N)]);
bineq = sparse([ones(N,1)*(-wmin); ones(N,1)*wmax]);
%%
%equality constraint
%Aeq: Matrix in linear equality constraints Aeq*x = beq
%beq: Vector in linear equality constraints Aeq*x = beq


LEFTMATRIX = zeros(2*N,N);
for i = 1:N
    LEFTMATRIX(2*i,i)=-h;
end

RIGHTMATRIX = zeros(2*N,2*N);
for i = 1:2*N
    if mod(i,2) == 1 %if odd
```

```
        RIGHTMATRIX(i,i)=1;
            if (i<2*N-1)%to prevent out of indexing
            RIGHTMATRIX(i+2,i)=-1;
            end
    else %if even
        if i<2*N-1
            RIGHTMATRIX(i,i)=1;
            RIGHTMATRIX(i+1,i)=-h*v;
            RIGHTMATRIX(i+2,i)=-1;
        elseif i<2*N
            RIGHTMATRIX(i,i)=1;
            RIGHTMATRIX(i+1,i)=-h*v;
        else %i=2*N
            RIGHTMATRIX(i,i)=1;
        end
    end
end


Aeq = [LEFTMATRIX RIGHTMATRIX];
beq = [y0+h*v*h0; h0; zeros(2*N-2,1)];
%%
%allocating memory for vectors
y=zeros(1,N+1);
head=zeros(1,N+1);
y(1)=y0;
head(1)=h0;
w=zeros(1,N+1);

for i=2:N+1 % must start at 2 since x(i-1)=x(1) for first iteration

    %update initial conditions
    y0=y(i-1);
    h0=head(i-1);

    %update constraints that depend on initial conditions
    beq = [y0+h*v*h0; h0; zeros(2*N-2,1)];

    %optimize
    [x, fval] = quadprog(H,f,Aineq,bineq,Aeq,beq);

    %input w0 value
    w(i-1)=x(1);

    %dynamics...y index starts at 1 (initial condition)
    y(i)=y(i-1)+h*v*head(i-1);
    head(i)=head(i-1)+h*w(i-1)+h*pi/25;
end

t=(0:N)*h;
subplot(3,1,1);
plot(t,y);
title('{yk} with Model Predictive Control');
xlabel('Time (s)')
ylabel('Position');
```

```
subplot(3,1,2);
plot(t, head);
title('{hk} with Model Predictive Control');
xlabel('Time (s)');
ylabel('Heading');

subplot(3,1,3);
plot(t, w);
title('Turn Rate with Model Predictive Control');
xlabel('Time (s)');
ylabel('w (rad/s)');
```

This plot takes a couple of minutes to compile! The plot of this is similar to the first plot without the disturbance factor. Both the position and heading converges to 0 and the robot follows its desired trajectory of a straight line. The difference between e) and f) is that in this case, there is an optimization step in each time-step reduces or remediates the disturbance – a “correction” factor.