# Matlab Appendix

**Q5**

|  |
| --- |
| function displayResults(hh, fsamp)  figure(1);  subplot(2,1,1)  [H,f] = freqz(hh,1,1024,fsamp);  semilogy(f,abs(H)), grid on  title('Frequency responce F\{h\_d[n]\}')  ylabel('|H\_d(\omega)|')  xlabel('Frequency[Hz]')    subplot(2,1,2)  stem(0:length(hh)-1, hh);  title('Impulse responce')  ylabel('h\_d[n]');  xlabel('n');    end |

|  |
| --- |
| function h=Q5    %% define LPF parameters  L = 500;  DeltaP = 0.1;  DeltaS = 0.001;    %% Equiriple design of LPF-using LP  wp = 0.26\*pi;  ws = 0.34\*pi;  wc = 0.30\*pi;  w = (0:L)\*pi/L;  %weighting matrix  S = diag(1/DeltaP\*(w<=wp) + 1/DeltaS\*(w>=ws));  d = (w<=wc)';%desired LPF frequency responce      for N = 5:2:101,  [delta, h] = EquirippleDesign(S, d);  displayResults(h, 2)  disp(['N: ' num2str(N)])  disp(['delta: ' num2str(delta)])  %Once delta is smaller than 1 we know that we satisfy the design  %constraints  if ( delta < 1 )  break;  end  end    %% Nested Functions  function [del, h] = EquirippleDesign(S, H\_d)  M = (N-1)/2;  C = [ones(L+1, 1) cos(w'\*[1:M])];    A = [ S\*C -ones(L+1,1);  -S\*C -ones(L+1,1);];  b = [S\*H\_d; -S\*H\_d];  f = [zeros(M+1,1); 1];    % solve linear program  options = optimoptions('linprog', 'Algorithm', 'simplex');  x = linprog(f, A, b, [], [], [], [], [], options);    del = x(end);  a = x(1:M+1);  % h = [0.5\*a(M+1:-1:2); a(1); 0.5\*a(2:M+1)];  T2 = aToh(N);  h = 0.5\*T2\*a;    end    end |

|  |
| --- |
| function T2 = aToh(N)  M = (N-1)/2;    T2 = eye(M+1);  T2 = [flipud(T2(2:end, :)); T2];  T2(M + 1, 1) = 2;  end |

**Q6+8**

|  |
| --- |
| function Q6and8(h)    %% define LPF parameters  L = 500;  DeltaP = 0.1;  DeltaS = 0.001;    %% Equiriple design of LPF-using LP  wp = 0.26\*pi;  ws = 0.34\*pi;  wc = 0.30\*pi;  w = (0:L)\*pi/L;  %weighting matrix  S = diag(1/DeltaP\*(w<=wp) + 1/DeltaS\*(w>=ws));  d = (w<=wc)';%desired LPF frequency responce    N = length(h);    T1 = fliplr(tril(ones(N)));  s = T1\*h;    figure;  stem(0:length(h)-1, s);  title('Step responce')  ylabel('s[n]');  xlabel('n');    %% Q8 - Time constraints  DeltaT = 0.05;      for N1 = 20:-1:1;  T2 = aToh(N);  T = T1(1:N1+1, :)\*T2;      [delta, h\_new] = EquirippleDesign(S, d, T);  displayResults(h, 2)  disp(['delta: ' num2str(delta)])  disp(['N1: ' num2str(N1)])    s\_new = T1\*h\_new;    figure(2);  stem(0:length(h\_new)-1, s, 'b');  hold on;  stem(0:length(h\_new)-1, s\_new, 'g');  hold off;  title('Step responce - time constraints')  ylabel('s[n]');  xlabel('n');  legend('Original step responce', 'Step responce - with constraints')  if ( delta < 1)  break;  end  end    %% Nested Functions  function [del, h] = EquirippleDesign(S, H\_d, T)  M = (N-1)/2;  C = [ones(L+1, 1) cos(w'\*(1:M))];    A = [ S\*C -ones(L+1,1);  -S\*C -ones(L+1,1);  T zeros(N1+1,1);  -T zeros(N1+1,1);];  v = 2\*DeltaT\*ones(N1+1, 1);  b = [S\*H\_d; -S\*H\_d; v; v];  f = [zeros(M+1,1); 1];    % solve linear program  options = optimoptions('linprog', 'Algorithm', 'simplex');  x = linprog(f, A, b, [], [], [], [], [], options);    del = x(end);  a = x(1:M+1);  % h = [0.5\*a(M+1:-1:2); a(1); 0.5\*a(2:M+1)];  T2 = aToh(N);  h = 0.5\*T2\*a;    end    end |

Q13

|  |
| --- |
| function alpha = ArmijoRule(f, A, b, x\_k, f\_k, gradf\_k, d\_k, sigma, beta, alpha\_0)  %INPUT:  % f - handle to function f()  % x\_k - x at current iteration  % f\_k - f(x\_k)  % gradk\_k - grad(f(x\_k))  % d\_k - can be -grad(f(x\_k)) or H^-1\*grad(f(x\_k))  % sigma - a bigger sigma make the search more conservative  % beta - smaller beta makes the line search converge faster at the expense  % of accuracy  %  %OUTPUT:  % alpha\_k - alpha\_k ~ arg min(f(x\_k + alpa\*d\_k))  %  %Armijo-Goldstein rule  %https://en.wikipedia.org/wiki/Backtracking\_line\_search    %Init  m = size(A, 1);  alpha = alpha\_0;  x = x\_k + alpha\*d\_k;  t = -sigma\*d\_k'\*gradf\_k;  k = 1;  %Iterate  while (f(x) - f\_k > sigma\*gradf\_k'\*(x - x\_k) || sum(A\*x-b<-eps) < m )%if point not strictly feasible keep on shrinking  alpha = beta\*alpha;%shrink alpha  % straight line  x = x\_k + alpha\*d\_k;  %%TODO check if is still interior point    k = k + 1;  if ( k >= 100 )  disp('warning: Armijo rule not converging, k> 100');  end  end |

|  |
| --- |
| function [x, Cost] = NewtonMethod(A, b, c, t, x0, maxIter, tol)  % Solves the following centering problem via Projected Newton:  %  % minimize t\*c^T\*x + phi(x)  % subject to Ax < b    %OUTPUT:  % x - Constrained optimal solution  % Cost - vector of decreasing cost  %PARAMS  sigma = 0.01;% For Armijo rule  beta = 0.5;% For Armijo rule  %Init  alpha\_0 = 1;% has to be 1 for Newton method to work    f = @(x)(t\*c'\*x -sum(log(b - A \* x)));  grad\_f = @(x)(t\*c + sum(bsxfun(@rdivide, A', (b - A \* x)'), 2));  hessian\_f = @(x)(A' \* diag(1./((b - A \* x).^2))\*A);    %iterate  xOld = inf(length(x0), 1);  x = x0;  dim = length(x0);  H = ones(dim); %#ok<NASGU>  d = zeros(dim, 1); %#ok<NASGU>  Costs = zeros(1, maxIter);  m = size(A, 1);    for k = 1:maxIter,  if ( norm(x-xOld)<tol\*norm(xOld) )  break;  end  gradf\_x = grad\_f(x);  H = hessian\_f(x);  % Newton step  d = -H\gradf\_x;    % a\_k ~ arg min(f + a\*d)  alpha\_k = ArmijoRule(f, A, b, x, f(x), grad\_f(x), d, sigma, beta, alpha\_0);    %update  xOld = x;  x = x + alpha\_k\*d;    if ( sum(A\*x < b) < m )  error('Point returned from newton step is not strictly feasible');  end    if (mod(k, 1000)==0)  disp(['Newton Iteration: ' num2str(k)])  end  end  disp(['Newton step converged at iteration ' num2str(k)]);  Cost = f(x); |

|  |
| --- |
| function [x, Cost] = LogBarrierMethod(A, b, c, t0, x0, mu, epsilon)    m = size(A, 1);    %check to see that the intial point is strictly feasible  if ( sum(A\*x0 < b) < m )  error('Initial point is infeasible')  end    t = t0;  x = x0;  maxIter = 10000;  tol = 1e-6;  OldCost = inf;    for k = 1:1000,  if (m/t < epsilon)  break;  end  % options = optimoptions(@fminunc,'GradObj','on','Hessian','on', ...  % 'MaxIter', maxIter, 'tolX', tol, 'MaxFunEvals', maxIter);%, 'DerivativeCheck', 'on');  % [x\_ref, Cost] = fminunc(@myfun, x, options);  [x, Cost] = NewtonMethod(A, b, c, t, x, maxIter, tol);  % disp(['Diff between reference my func: ' num2str(norm(x-x\_ref))])  disp(['LP Cost is now: ' num2str(c'\*x)]);  if ( sum(A\*x < b) < m )  error('Point returned from newton step is not strictly feasible');  end  if (abs(Cost-OldCost) < tol\*Cost)  disp('Cost not changing')  break;  end    t = t\*mu;  end    Cost = c'\*x;    %% Nested function - for fminunc and verification  function [f\_x, g, H] = myfun(x)  f\_x = t\*c'\*x - sum(log(b - A \* x)); % Cost function  g = t\*c + sum(bsxfun(@rdivide, A', (b - A \* x)'), 2);  H = A' \* diag(1./((b - A \* x).^2))\*A;  end    end |

|  |
| --- |
| function [x, Cost] = LogBarrierSolver(A, b, c, t0, mu, epsilon)      %% Phase 1 - Find strictly feasible starting point    gamma\_0 = max(-b)+ 1;  AA = [A -ones(size(A,1), 1);  zeros(1, size(A,2)) -1;];%lower bound constraint on gamma -> -gamma<=1 -> gamma > -1    bb = [b; 1];  cc = [zeros(size(A,2), 1); 1];  xx\_0 = [zeros(size(A,2), 1); gamma\_0];    xx\_feas = LogBarrierMethod(AA, bb, cc, t0, xx\_0, mu, epsilon);    % For debug  % options = optimoptions('linprog', 'Algorithm', 'simplex');  % xx\_feas\_ref = linprog(cc, AA, bb, [], [], [], [], xx\_0, options);    %% Phase 2  x0 = xx\_feas(1:end-1);  [x, Cost] = LogBarrierMethod(A, b, c, t0, x0, mu, epsilon); |

|  |
| --- |
| function [A, b, c] = FirstFIRProblem(wp, ws, wc, L, N, DeltaP, DeltaS)    w = (0:L)\*pi/L;  %weighting matrix  S = diag(1/DeltaP\*(w<=wp) + 1/DeltaS\*(w>=ws));  d = (w<=wc)';%desired LPF frequency responce      M = (N-1)/2;  C = [ones(L+1, 1) cos(w'\*(1:M))];    A = [ S\*C -ones(L+1,1);  -S\*C -ones(L+1,1);];  b = [S\*d; -S\*d];  c = [zeros(M+1,1); 1]; |

|  |
| --- |
| function [A, b, f] = SecondFIRProblem(wp, ws, wc, L, N, N1, DeltaP, DeltaS, DeltaT)    M = (N-1)/2;    %% Frequency  w = (0:L)\*pi/L;    %weighting matrix  S = diag(1/DeltaP\*(w<=wp) + 1/DeltaS\*(w>=ws));  d = (w<=wc)';%desired LPF frequency responce    %% Get time constraints  T1 = fliplr(tril(ones(N)));  T2 = aToh(N);  T = T1(1:N1+1, :)\*T2;    %% Calc LP pramaters A, b, f    C = [ones(L+1, 1) cos(w'\*(1:M))];    A = [ S\*C -ones(L+1,1);  -S\*C -ones(L+1,1);  T zeros(N1+1,1);  -T zeros(N1+1,1);];  v = 2\*DeltaT\*ones(N1+1, 1);  b = [S\*d; -S\*d; v; v];  f = [zeros(M+1,1); 1]; |

|  |
| --- |
| %Q13    %% define LPF parameters  L = 500;  DeltaP = 0.1;  DeltaS = 0.001;  DeltaT = 0.05;  N = 47;  N1 = 18;    %% Equiriple design of LPF-using LP  wp = 0.26\*pi;  ws = 0.34\*pi;  wc = 0.30\*pi;  w = (0:L)\*pi/L;    %% Lob Barrier Parameters  t0 = 500;  mu = 15;  epsilon = 1e-6;    %% First FIR design problem  [A, b, c] = FirstFIRProblem(wp, ws, wc, L, N, DeltaP, DeltaS);    tic  x = LogBarrierSolver(A, b, c, t0, mu, epsilon);  disp('Time for log Barrier')  toc    tic;  options = optimoptions('linprog', 'Algorithm', 'simplex');  x\_ref = linprog(c, A, b, [], [], [], [], [], options);  disp('Time for simplex(linprog)')  toc    disp('Difference between linprog and my Log-Barrier Method:')  norm(x-x\_ref)    [h, del] = xToh(x, N);  [h\_ref, ~] = xToh(x\_ref, N);    figure;  stem(0:length(h)-1, h);  hold on;  stem(0:length(h\_ref)-1, h\_ref, 'g');  hold off  legend('Log-Barrier Method', 'Simplex(linprog)')  title('Impulse responce - First problem')  ylabel('h[n]');  xlabel('n');    %% Second FIR design problem with time constraints  [A, b, f] = SecondFIRProblem(wp, ws, wc, L, N, N1, DeltaP, DeltaS, DeltaT);      tic  xx\_feas = LogBarrierSolver(A, b, c, t0, mu, epsilon);  disp('Time for log Barrier')  toc    tic;  x\_ref = linprog(c, A, b, [], [], [], [], [], options);  disp('Time for simplex(linprog)')  toc    disp('Difference between linprog and my Log-Barrier Method:')  norm(x-x\_ref)    [h, del] = xToh(x, N);  [h\_ref, ~] = xToh(x\_ref, N);    figure;  stem(0:length(h)-1, h);  hold on;  stem(0:length(h\_ref)-1, h\_ref, 'g');  hold off  legend('Log-Barrier Method', 'Simplex(linprog)')  title('Impulse responce - With time constraints')  ylabel('h[n]');  xlabel('n'); |