Linear Programming (LP)

- If we can get our filter or window design problems in the form
  \[
  \begin{align*}
  \text{minimize} & \quad f^T x \\
  \text{subject to} & \quad A_{eq} x = b_{eq}, \\
  & \quad A x \leq b
  \end{align*}
  \]
  where \( x, f \in \mathbb{R}^N, b \in \mathbb{R}^M, A \) is \( M \times N \), etc., then we are done.

- The `linprog` function in Matlab Optimization Toolbox solves LP problems.
Matlab’s LINPROG

>> help linprog

LINPROG Linear programming.
X=LINPROG(f,A,b) solves the linear programming problem:

\[ \begin{align*}
\min & \quad f'x \\
\text{subject to:} & \quad A x \leq b \\
\end{align*} \]

X=LINPROG(f,A,b,Aeq,beq) solves the problem above while additionally satisfying the equality constraints Aeq*x = beq.

X=LINPROG(f,A,b,Aeq,beq,lb,ub) defines a set of lower and upper bounds on the design variables, X, so that the solution is in the range LB <= X <= UB. Use empty matrices for LB and UB if no bounds exist. Set LB(i) = -Inf if X(i) is unbounded below; set UB(i) = Inf if X(i) is unbounded above.

X=LINPROG(f,A,b,Aeq,beq,lb,ub,x0) sets the starting point to X0. This option is only available with the active-set algorithm. The default interior point algorithm will ignore any non-empty starting point.

X=LINPROG(f,A,b,Aeq,beq,lb,ub,x0,options) minimizes with the default optimization parameters replaced by values in the structure OPTIONS, an argument created with the OPTIMSET function. See OPTIMSET for details. Use options are Display, Diagnostics, TolFun, LargeScale, MaxIter. Currently, only ‘final’ and ‘off’ are valid values for the parameter Display when LargeScale is ‘off’ (‘iter’ is valid when LargeScale is ‘on’).

[X,fval,exitflag,output] = LINPROG(f,A,b) returns the value of the objective function at X: FVAL = f'*X.

[X,fval,exitflag] = LINPROG(f,A,b) returns EXITFLAG that describes the exit condition of LINPROG.
If EXITFLAG is:
> 0 then LINPROG converged with a solution X.
0 then LINPROG reached the maximum number of iterations without converging.
< 0 then the problem was infeasible or LINPROG failed.

[X,fval,exitflag,lambda] = LINPROG(f,A,b) returns a structure OUTPUT with the number of iterations taken in OUTPUT.iterations, the type of algorithm used in OUTPUT.algorithm, the number of conjugate gradient iterations (if used) in OUTPUT.cgiterations.

[X,fval,exitflag,OUTPUT,lambda] = LINPROG(f,A,b) returns the set of Lagrangian multipliers LAMBDA, at the solution: LAMBDA.ineqlin for the linear inequalities A, LAMBDA.eqlin for the linear equalities Aeq, LAMBDA.lower for LB, and LAMBDA.upper for UB.

NOTE: the LargeScale (the default) version of LINPROG uses a primal-dual method. Both the primal problem and the dual problem must be feasible for convergence. Infeasibility messages of either the primal or dual, or both, are given as appropriate. The primal problem in standard form is
\[ \begin{align*}
\min & \quad f'x \\
\text{subject to:} & \quad A x = b, \ x \geq 0. \\
\end{align*} \]
The dual problem is
\[ \begin{align*}
\max & \quad b'y \\
\text{subject to:} & \quad A'y + s = f, \ s \geq 0. \\
\end{align*} \]
Using \texttt{cvx} in place of \texttt{linprog}

The \textit{disciplined convex optimization toolbox} \texttt{cvx}ootnote{http://cvxr.com/cvx/download/} is available from Prof. Stephen Boyd’s Stanford/EE website, and it should be installed on CCRMA machines. To use \texttt{cvx}, build the matrices exactly as for \texttt{linprog}, but instead of calling \texttt{linprog}, use the following code:

\begin{verbatim}
f=f';
cvx_begin
    variable x(N)
    minimize f'*x
    subject to
        A*x<=b
        Aeq*x==beq
cvx_end
\end{verbatim}

See also the book\footnote{http://www.stanford.edu/~boyd/cvxbook/}

---

Formulation of Chebyshev Window Design as an LP

What we want:

1. \textit{Symmetric} zero-phase window.
2. Window samples to be \textit{positive}.
   \[ w(n) \geq 0 \quad \text{for} \quad -\frac{M-1}{2} \leq n \leq \frac{M-1}{2} \equiv L \]
3. Transform to be \textit{unity} at DC.
   \[ W(0) = 1 \]
4. Transform to be within \([-\delta, \delta]\) in the stop-band.
   \[ -\delta \leq W(\omega) \leq \delta \quad \text{for} \quad \omega_{sb} \leq \omega \leq \pi \]
5. And \(\delta\) to be small.
**Symmetric Window Constraint**

Because we are designing a zero-phase window, use only the positive-time part \( h(n) \):

\[
h(n) = w(n) \quad n \geq 0
\]

\[
w(n) = \begin{cases} h(n) & n \geq 0 \\ h(-n) & n < 0 \end{cases}
\]

---

**Positive Window Sample Constraint**

For each window sample, \( h(n) \geq 0 \), or,

\[
-h(n) \leq 0.
\]

Stacking inequalities for all \( n \),

\[
\begin{bmatrix}
-1 & 0 & \cdots & 0 & 0 \\
0 & -1 & 0 & \cdots & 0 \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
0 & 0 & -1 & 0 & \cdots \\
0 & 0 & \cdots & 0 & -1
\end{bmatrix}
\begin{bmatrix}
h(0) \\
h(1) \\
\vdots \\
h(L-1) \\
h(L)
\end{bmatrix}
\leq
\begin{bmatrix}
0 \\
0 \\
\vdots \\
0 \\
0
\end{bmatrix}
\]

or

\[
-Ih \leq 0.
\]
DC Constraint

The DTFT at frequency $\omega$ is given by

$$W(\omega) = \sum_{n=-L}^{L} w(n) e^{-j\omega n}. \quad L = \frac{M-1}{2}$$

Using the symmetry,

$$W(\omega) = h(0) + 2 \sum_{n=1}^{L} h(n) \cos(n\omega)$$

$$= \begin{bmatrix} 1 & 2 \cos(\omega) & \cdots & 2 \cos(L\omega) \end{bmatrix} \begin{bmatrix} h(0) \\ h(1) \\ \vdots \\ h(L) \end{bmatrix}$$

$$= d(\omega)^T h.$$ 

So $W(0) = 1$ can be expressed as

$$d(0)^T h = 1.$$ 

Side-Lobe Specification

Likewise, the side-lobe specification can be enforced at frequencies $\omega_i$ in the stop-band.

$$-\delta \leq d(\omega_i)^T h \leq \delta$$

or

$$\begin{cases} -d(\omega_i)^T h - \delta \leq 0 \\ d(\omega_i)^T h - \delta \leq 0 \end{cases}$$

where,

$$\omega_{sb} \leq \omega_1, \omega_2, \ldots, \omega_K \leq \pi.$$ 

We need $K \gg L$ to obtain many frequency samples per side lobe.
Stacking inequalities for all $\omega_i$,

\[
\begin{bmatrix}
-d(\omega_1)^T \\
\vdots \\
-d(\omega_K)^T \\
d(\omega_1)^T \\
\vdots \\
d(\omega_K)^T
\end{bmatrix} h +
\begin{bmatrix}
-\delta \\
\vdots \\
-\delta \\
-\delta \\
\vdots \\
-\delta
\end{bmatrix}
\leq 0
\]

or

\[
\begin{bmatrix}
-d(\omega_1)^T & -1 \\
\vdots & \vdots \\
-d(\omega_K)^T & -1 \\
d(\omega_1)^T & -1 \\
\vdots & \vdots \\
d(\omega_K)^T & -1
\end{bmatrix}
\begin{bmatrix}
h \\
\delta
\end{bmatrix}
\leq 0.
\]

\textit{i.e.,}

\[
A_{sb} \begin{bmatrix} h \\ \delta \end{bmatrix} \leq 0.
\]

\textbf{LP Standard Form}

Now gather all of the constraints to form an LP problem:

\[
\text{minimize } \begin{bmatrix} 0 & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} h \\ \delta \end{bmatrix}
\]

subject to

\[
\begin{bmatrix} d(0)^T & 0 \end{bmatrix} \begin{bmatrix} h \\ \delta \end{bmatrix} = 1
\]

and

\[
\begin{bmatrix} -I & 0 \end{bmatrix} A_{sb} \begin{bmatrix} h \\ \delta \end{bmatrix} \leq 0
\]

where the optimization variables are $\begin{bmatrix} h \\ \delta \end{bmatrix}$.

- This should produce a window that is optimal in the Chebyshev sense over the chosen frequency samples, as shown on the next page.

- If the chosen frequency samples include all the extremal frequencies (frequencies of maximum error in the DTFT of the window), then the unique Chebyshev window for the specified main-lobe width must be obtained.
Remez Exchange Algorithm

- The Remez multiple exchange algorithm works by moving the frequency samples each iteration to points of maximum error (on a denser grid).
- Remez iterations could be added to our formulation here as well, enabling the use of many fewer frequency samples ($K \approx L$).
- The Remez multiple exchange algorithm (firpm [formerly remez] in the Matlab Signal Processing Toolbox) is normally faster than a linear programming formulation, which can be regarded as a single exchange method [Rabiner and Gold 1975, p. 140] on a larger frequency grid.
- Another reason for the speed of firpm is that it solves the following equations non-iteratively for the filter exhibiting the desired error alternation over the current set of extremal frequencies:

$$
\begin{bmatrix}
H(\omega_1) \\
H(\omega_2) \\
\vdots \\
H(\omega_K)
\end{bmatrix} = 
\begin{bmatrix}
1 & 2\cos(\omega_1) & \cdots & 2\cos(\omega_1L) & \frac{1}{W(\omega_1)} \\
1 & 2\cos(\omega_2) & \cdots & 2\cos(\omega_2L) & \frac{1}{W(\omega_2)} \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
1 & 2\cos(\omega_K) & \cdots & 2\cos(\omega_KL) & \frac{1}{W(\omega_K)}
\end{bmatrix}
\begin{bmatrix}
h_0 \\
h_1 \\
\vdots \\
h_L \\
\delta
\end{bmatrix}
$$

or, in terms of our previous notation,

$$
H(\omega_k) = d(\omega_k)^T h + \delta \cdot \frac{(-1)^k}{W(\omega_k)}
$$

where $W(\omega_k)\delta$ is the weighted ripple amplitude at frequency $\omega_k$. ($W(\omega_k)$ is an arbitrary ripple weighting function.) Note that the desired frequency response amplitude $H(\omega_k)$ is also arbitrary (but real) at each frequency sample.

Convergence of Remez Exchange

- In theory, $\delta$ is guaranteed to increase monotonically each iteration, ultimately converging to its optimal value. This value is reached when all the extremal frequencies $\omega_k^*$ are found.
- In practice, numerical round-off error may cause $\delta$ not to increase monotonically. When this is detected, the algorithm halts and reports a failure to converge.
- Convergence failure is common in practice for FIR filters having more than 300 or so taps and stringent design specifications (such as very narrow pass-bands).
- Further details on Remez exchange are given in [Rabiner and Gold 1975, p. 136].
• As a result of the non-iterative internal solution on each iteration, \texttt{firpm} cannot be used when additional constraints are added, such as those to be discussed in the following sections. In such cases, a more general LP solver such as \texttt{linprog} or \texttt{cvx} must be used.

• Recent advances in convex optimization enable faster solution of much larger problems (see EE364 A,B)
Monotonicity Constraint

We can constrain the positive-time part of the window to be monotonically decreasing.

\[ \Delta h_i = h(i+1) - h(i) \leq 0 \quad i = 1, \ldots, L - 1 \]

In matrix form,

\[
\begin{bmatrix}
-1 & 1 & 0 \\
-1 & 1 & 1 \\
0 & \cdots & -1 & 1
\end{bmatrix}
\begin{bmatrix}
h_1 \\
h_2 \\
\vdots \\
h_L
\end{bmatrix}
\leq 0,
\]

or,

\[ D h \leq 0. \]
**L-Infinity Norm of Derivative Objective**

We can add a *smoothness objective* by adding $L_\infty$-norm of the first-order difference to the objective function.

\[
\text{minimize } \delta + \eta \| \Delta h \|_\infty.
\]

- $L_\infty$-norm only cares about the *maximum difference*.
- Large $\eta$ means we put more weight on the smoothness than the side-lobe level.

Let $\sigma \triangleq \| \Delta h \|_\infty$ and set up the inequality constraints

\[-\sigma \leq \Delta h_i \leq \sigma \quad i = 1, \ldots, L - 1.
\]

In matrix form:

\[
\begin{bmatrix}
-D \\
D
\end{bmatrix} h - \sigma \mathbf{1} \leq 0.
\]

The objective function is then

\[
\text{minimize } \delta + \eta \sigma.
\]
Twenty times the Chebyshev norm of \( \text{diff}(h) \) added to the objective function to be minimized \((\eta = 20)\):

![Graphs showing window, tap-tap difference, and window transform.]

### L-One Norm of Derivative Objective

Another way to add a smoothness constraint is to add the \( L_1 \)-norm of the derivative to the objective.

\[
\text{minimize} \quad \delta + \eta \| \Delta h \|_1.
\]

- The \( L_1 \) norm is sensitive to all the derivatives, not just the largest.

In LP, set up the inequality constraints

\[
-\tau_i \leq \Delta h_i \leq \tau_i \quad i = 1, \ldots, L - 1,
\]

or, in matrix form,

\[
\begin{bmatrix}
-D \\
D
\end{bmatrix} h - \begin{bmatrix}
\tau \\
\tau
\end{bmatrix} \leq 0.
\]

The objective function becomes

\[
\text{minimize} \quad \delta + \eta 1^T \tau.
\]
$L_1$ norm of $\text{diff}(h)$ added to the objective function ($\eta = 1$):

Six times the $L_1$ norm of $\text{diff}(h)$ added to the objective function ($\eta = 6$):