11.1 The Discrete Cosine Transform
Discrete Cosine Transform

\[ C = \sqrt{\frac{2}{n}} \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \cos \frac{\pi}{2n} & \cos \frac{3\pi}{2n} & \ldots & \cos \frac{2(n-1)\pi}{2n} \\ \cos \frac{3\pi}{2n} & \cos \frac{6\pi}{2n} & \ldots & \cos \frac{2(n-2)\pi}{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \cos \frac{(n-1)\pi}{2n} & \cos \frac{(n-1)3\pi}{2n} & \ldots & \cos \frac{(n-1)(n-2)\pi}{2n} \end{bmatrix} \]

\[ c_{i,j} = \sqrt{\frac{2}{n}} a_i \cos \left( \frac{i(2j+1)\pi}{2n} \right) \quad \text{where} \quad a_i = \begin{cases} \frac{1}{\sqrt{2}} & \text{if} \quad i = 0 \\ 1 & \text{if} \quad i = 1, \ldots, n-1 \end{cases} \]

For \( \mathbf{x} = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{pmatrix} \), the D.C.T. of \( \mathbf{x} \) is

\[ y = C \mathbf{x} \]
Claim: columns of $C^T$ are (unit) eigenvectors of a real, symmetric matrix.

\[
C^T = \sqrt{\frac{2}{n}} \begin{bmatrix}
1/\sqrt{2} & \cos \frac{\pi}{2n} & \cos \frac{2\pi}{2n} & \cdots & \cos \frac{(n-1)\pi}{2n} \\
1/\sqrt{2} & \cos \frac{3\pi}{2n} & \cos \frac{6\pi}{2n} & \cdots & \cos \frac{(n+1)\pi}{2n} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1/\sqrt{2} & \cos \frac{(2n-1)\pi}{2n} & \cos \frac{(2n-2)\pi}{2n} & \cdots & \cos \frac{(n-1)(2n-1)\pi}{2n} \\
1/\sqrt{2} & \cos \frac{(2n)\pi}{2n} & \cos \frac{(2n-1)\pi}{2n} & \cdots & \cos \frac{(n-1)(2n-2)\pi}{2n}
\end{bmatrix}
\]

\[
\begin{bmatrix}
1 & -1 & 0 & 0 & 0 \\
-1 & 2 & -1 & 0 & 0 \\
0 & -1 & 2 & -1 & 0 \\
0 & 0 & -1 & 2 & -1 \\
0 & 0 & 0 & -1 & 1
\end{bmatrix}
\]

\[
\text{The columns of } C^T \text{ form an orthogonal set of vectors.}
\]
\[ \therefore \ C^T \text{ is orthogonal matrix; } \]
\[ (C^T)^{-1} = (C^T)^T = C \]

Take inverses:
\[ C^T = ((C^T)^T)^{-1} = C^{-1} \]

\[ \therefore \ C \text{ itself is an orthogonal matrix.} \]

Let \( t_0 = \frac{1}{\sqrt{n}} \), \( t_0 = 0 \)
\[ t_1 = \frac{\sqrt{2}}{\sqrt{n}} \cos \left( \frac{2(t_1+1)\pi}{2n} \right) \quad t_1 = 1 \]
\[ t_2 = \frac{\sqrt{2}}{\sqrt{n}} \cos \left( \frac{2(t_2+1)\pi}{2n} \right) \quad t_2 = 2 \]
\[ t_{n-1} = \frac{\sqrt{2}}{\sqrt{n}} \cos \left( \frac{(n-1)(2t+1)\pi}{2n} \right) \quad t_{n-1} = n-1 \]
Then Theorem 10.9 applies:

\[ F(t) = \sum_{k=0}^{n-1} y_k f_k(t) \] satisfies \( F(t_j) = x_j, j = 0, \ldots, n-1 \)

where \( y = A \alpha = \begin{bmatrix} f_0(t_0) & f_0(t_1) & \cdots & f_0(t_{n-1}) \\ f_1(t_0) & f_1(t_1) & \cdots & f_1(t_{n-1}) \\ \vdots & \vdots & \ddots & \vdots \\ f_{n-1}(t_0) & f_{n-1}(t_1) & \cdots & f_{n-1}(t_{n-1}) \end{bmatrix} \alpha \)

Here, \( A = C \); and \( C \) is an orthogonal matrix

\[ y = C \alpha \] is the Discrete Cosine Transform of \( \alpha = \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_{n-1} \end{bmatrix} \)

Then let \( F(t) = P_n(t) \)

\[ P_n(t) = \frac{1}{\sqrt{n}} y_0 + \sqrt{\frac{2}{n}} \sum_{k=1}^{n-1} y_k \cos \left( \frac{k(2t+1)\pi}{2n} \right) \]

satisfies \( P(t_j) = x_j, j = 0, \ldots, n-1 \)
11.2 Two-dimensional DCT and image compression
Recall \( \text{Discrete Cosine Transform} \).

\[
C = \sqrt{\frac{2}{n}} \begin{bmatrix}
\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\
\cos \frac{\pi}{2n} & \cos \frac{3\pi}{2n} & \cdots & \cos \frac{(2n-1)\pi}{2n} \\
\cos \frac{2\pi}{2n} & \cos \frac{6\pi}{2n} & \cdots & \cos \frac{2(2n-1)\pi}{2n} \\
\vdots & \vdots & \ddots & \vdots \\
\cos \frac{(n-1)\pi}{2n} & \cos \frac{(n-1)3\pi}{2n} & \cdots & \cos \frac{\pi(n-1)(2n-1)}{2n}
\end{bmatrix}
\]

\[c_i = \frac{1}{\sqrt{n}} \cos \left( \frac{i(2j+1)\pi}{2n} \right) \text{ where } a_i = \begin{cases} \frac{1}{\sqrt{2}} & \text{if } i = 0 \\ 1 & i = 1, \ldots, n-1 \end{cases} \]

For \( x = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{pmatrix} \), the D.C.T. of \( x \) is

\[
y = C x : \\
y_k = \sum_{i=0}^{n-1} C_{ki} x_i
\]

Note: summation from 0 to \( n-1 \)
Consider 2D data \( \{ x_{ij} \} \), \( i = 0, \ldots, n-1 \)
\( j = 0, \ldots, n-1 \)

We transform \( \{ x_{ij} \} \) into \( \{ y_{kl} \} \), the 2D D.C.T.

Define:
\[
y_{kl} = \sum_{j=0}^{n-1} C_{kj} \left( \sum_{i=0}^{n-1} C_{li} x_{ij} \right)
\]

But:
\[
y_{kl} = \sum_{j=0}^{n-1} \sum_{i=0}^{n-1} C_{kj} C_{li} x_{ij}
\]

\[
= \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} C_{ki} C_{lj} x_{ij}
\]

doesn't matter if transform \( i \rightarrow k \) first then \( j \rightarrow l \)

or \( j \rightarrow l \) first, then \( i \rightarrow k \)
\[ y_{kl} = \sum_{c=0}^{n-1} c_{ki} \left( \sum_{j=0}^{n-1} k_{ij} c_{lj} \right) \]

\[ = \sum_{c=0}^{n-1} c_{ki} \sum_{j=0}^{n-1} k_{ij} \left( C^T \right)_{jl} \]

\[ = (C \left( X C^T \right))_{kl} \]

\[ Y = C X C^T \text{ is the 2D D.C.T. matrix,} \]

\[ \begin{pmatrix} y_{00} & y_{01} & \cdots & y_{0n-1} \\ y_{10} & y_{11} & \cdots & y_{1n-1} \\ \vdots & \vdots & \ddots & \vdots \\ y_{n-0} & y_{n-11} & \cdots & y_{nn-1} \end{pmatrix} = C \begin{pmatrix} x_{00} & x_{01} & \cdots & x_{0n-1} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n-0} & x_{n-11} & \cdots & x_{nn-1} \end{pmatrix} \]
Start with \( Y = CXC^T \); find \( X \)?

- \( X \) on left: \( C^TY = C^TCC^T \)
  \[ \text{I since } C \text{ is orthogonal} \]

- \( X \) on right: \( C^TYC = XX^TC = X \)

The formula \( X = C^TYC \) is the inverse 2D-D.C.T.

Recall \( c_{ij} = \sqrt{\frac{1}{n}} a_i \cos \left( \frac{i(2j+1))\pi}{2n} \right) \quad i = 0, \ldots, n-1 \)

\( a_0 = \frac{1}{\sqrt{2}}, \quad a_1 = a_2 = \ldots = 1. \)
Write out $X = C^T Y C$:

$$
X_{i,j} = \frac{2}{n} \sum_{k=0}^{n-1} \sum_{l=0}^{n-1} y_{k,e} a_k a_e \cos \left( \frac{k(2i+1)\pi}{2n} \right) \cos \left( \frac{l(2j+1)\pi}{2n} \right)
$$

Let $P_n(s,t) = \frac{2}{n} \sum_{k=0}^{n-1} \sum_{l=0}^{n-1} y_{k,e} a_k a_e \cos \left( \frac{k(2s+1)\pi}{2n} \right) \cos \left( \frac{l(2t+1)\pi}{2n} \right)$.

Let $s_i = i, \quad t_j = j, \quad i,j = 0, \ldots, n-1$.

Then: $P_n(s,t)$ interpolates the data $x(s_i, t_j, X_{i,j})$

$s, j = 0, \ldots, n-1$

This is using the 2D DCT to find the coefficients $\{y_{k,e}\}$ of a trig. poly. $P_n(s,t)$, that interpolates the given data.
Suppose represent \( \{ \kappa_{ij} \} \) as follows:

\[
\begin{array}{c|ccc|c}
j = 0 & j = 1 \\
\hline
1 = 0 & \kappa_{00} & \kappa_{01} & \kappa_{02} & \kappa_{03} & \kappa_{0,n-1} \\
1 = 1 & \kappa_{10} & \kappa_{11} & \kappa_{12} \\
1 = 1 & \kappa_{n,0} & \kappa_{n,1} & \kappa_{n,2} & \kappa_{n,n-1}
\end{array}
\]
11.2.3 Quantization
Rei: DCT's:

For cases \( n = 2 \) & \( n = 4 \),

simple formulas for \( C, \ C^T \):

\( n = 2: \)

\[
C = \sqrt{\frac{2}{n}} \begin{pmatrix}
\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\
\cos \frac{\pi}{4} & \cos \frac{3\pi}{4}
\end{pmatrix}
\]

\[
= \frac{1}{\sqrt{2}} \begin{pmatrix}
1/\sqrt{2} & 1/\sqrt{2} \\
1/\sqrt{2} & -1/\sqrt{2}
\end{pmatrix}
\]

\( n = 4 \)

\[
C = \frac{1}{\sqrt{2}} \begin{pmatrix}
\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\
\cos \frac{\pi}{8} & \cos \frac{3\pi}{8} & \cos \frac{5\pi}{8} & \cos \frac{7\pi}{8}
\end{pmatrix}
\]

where \( a = \frac{1}{2} \), \( b = \frac{1}{\sqrt{2}} \cos \frac{\pi}{8} \), \( c = \frac{1}{\sqrt{2}} \cos \frac{3\pi}{8} \).

Makes exercises for \( n = 2 \), \( n = 4 \) "simple".
Byte: 8 bits of storage
patterns from 00000000, 00000001, 00000010, 00000011, ... up to 11111111.

say, integers from 0 to 255

If given the number "32" say, can store
in 8 bits, as 00100000 using all 8 bits.
Can save on bits, if can limit the range of values
to be stored.

leads to: Huffman encoding:

fewer bits for smaller numbers (more probable)
more bits for larger (i.e. less probable).
Idea: use fewer bit patterns to store information in higher frequency terms. Than (as compared with lower freq. terms).

"The" standard quantization matrix is

\[
Q_1 = \begin{bmatrix}
8 & 16 & 24 & 32 & 40 & 48 & 56 & 64 \\
16 & 24 & 32 & 40 & 48 & 56 & 64 & 72 \\
32 & 40 & 48 & 56 & 64 & 72 & 80 & 88 \\
40 & 48 & 56 & 64 & 72 & 80 & 88 & 96 \\
56 & 64 & 72 & 80 & 88 & 96 & 104 & 112 \\
64 & 72 & 80 & 88 & 96 & 104 & 112 & 120
\end{bmatrix}
\]

\[p_{508}\]

\[Q_2 = 2Q_1,\]

\[Q_4 = 4Q_1.\]
Define

\[ Y_Q = \begin{bmatrix}
\text{round}(\frac{y_{10}}{8}) & \text{round}(\frac{y_{11}}{16}) \\
\text{round}(\frac{y_{20}}{64}) & \text{round}(\frac{y_{21}}{72}) \\
\text{round}(\frac{y_{30}}{128}) & \text{round}(\frac{y_{31}}{144}) \\
\end{bmatrix}
\]

To restore \( Y \) from \( Y_Q \), use

\[ Y = \begin{bmatrix}
8y_{10}^Q & 16y_{11}^Q & 32y_{20}^Q & \cdots & 64y_{21}^Q \\
16y_{30}^Q & 24y_{31}^Q \\
\end{bmatrix} \]
12.1 Power iteration methods
Eigenvalues, powering, inverse powering.

Eigenvalues \( \lambda \) satisfy
\[
A \hat{\nu} = \lambda \hat{\nu}, \ \text{some} \ \hat{\nu} \neq \vec{0}.
\]

"shift": \( \hat{\nu} \) is \( \hat{\nu} = d \hat{\nu} \)
then \( (A - sI) \hat{\nu} = A \hat{\nu} - s \hat{\nu} = d \hat{\nu} - s \hat{\nu} \)
\[
= (d - s) \hat{\nu}.
\]

"shift" by \( s \).

Qf.: \( d_1, d_2, \ldots, d_n \) are the eigenvalues of \( A \)

\[
0 \quad d_2 \quad d_1 \quad d_1
\]

\( d_1 \) is dominant (for \( A \)).

Eigenvalues of \( A' = A - sI \).

\[
\left\{
\begin{array}{l}
d_1' = d_1 + s \\
d_2' = d_2 + s \\
d_3' = d_3 + s
\end{array}
\right.
\]

\( d_3 ' > d_2 ' > d_1 ' \) so: \( d_3 ' \) is dominant.

Note: \( |d_3'| > |d_2'| > |d_1'| \).
Suppose \( A \tilde{u}_k = \delta_k \tilde{u}_k \) for \( k = 1, \ldots, n \) and \( \delta_k \neq 0 \) for any \( k \).

Then, consider \( A \tilde{v} = \tilde{u} \).

Apply \( A^{-1} \):

\[
A^{-1} A \tilde{v} = \lambda A^{-1} \tilde{v}
\]

\[
\tilde{v} = \lambda A^{-1} \tilde{v}
\]

\[
A^{-1} \tilde{v} = (\frac{1}{\lambda}) \tilde{v}
\]

---

**eigenvalues of** \( A \)

\[
0 \quad \lambda_3 \quad \lambda_2 \quad \lambda_1
\]

---

**eigenvalues of** \( A^{-1} \)

\[
0 \quad \frac{1}{\lambda_3} \quad \frac{1}{\lambda_2} \quad \frac{1}{\lambda_1}
\]

Note: \( |\frac{1}{\lambda_3}| > |\frac{1}{\lambda_2}| > |\frac{1}{\lambda_1}| \), so \( (\frac{1}{\lambda_3}) \) is dominant (for \( A^{-1} \)).

Can recover: \( \lambda_3 = \frac{1}{(1/\lambda_3)} \)
Basic powering to approximate dominant eigenvalue.

Start with some $x^0$.

Form

$$x^{(0)} = c_1 \tilde{v}_1 + c_2 \tilde{v}_2 + \ldots + c_n \tilde{v}_n.$$ 

Usually, $x^{(0)} = c_1 \tilde{v}_1 + c_2 \tilde{v}_2 + \ldots + c_n \tilde{v}_n$.

Form

$$x^{(1)} = A x^{(0)} = c_1 A \tilde{v}_1 + c_2 A \tilde{v}_2 + \ldots + c_n A \tilde{v}_n$$

$$= c_1 d_1 \tilde{v}_1 + c_2 d_2 \tilde{v}_2 + \ldots + c_n d_n \tilde{v}_n$$

$$x^{(2)} = A x^{(1)} = c_1 d_1^2 \tilde{v}_1 + c_2 d_2^2 \tilde{v}_2 + \ldots + c_n d_n^2 \tilde{v}_n$$

Assume: $|d_1| > |d_2|$, $|d_3|$, ...

$$= d_1^2 \left( c_1 \tilde{v}_1 + c_2 \left( \frac{d_2}{d_1} \right)^2 \tilde{v}_2 + \ldots + c_n \left( \frac{d_n}{d_1} \right)^2 \tilde{v}_n \right)$$

If we say

$d_1$ is dominant

$$x^{(k)} = A^k x^{(0)} = d_1^k \left( c_1 \tilde{v}_1 + c_2 \left( \frac{d_2}{d_1} \right)^k \tilde{v}_2 + \ldots + c_n \left( \frac{d_n}{d_1} \right)^k \tilde{v}_n \right)$$

$k = 1, 2, 3, \ldots$

$$\left| \frac{d_2}{d_1} \right| < 1, \ldots, \left| \frac{d_n}{d_1} \right| < 1$$

$$\left| \frac{d_2}{d_1} \right|^k \to 0, \ldots, \left| \frac{d_n}{d_1} \right|^k \to 0$$
To combine these techniques:

\[ \begin{array}{c}
0 \quad d_3 \quad d_2 \quad d_1 \\
\text{eigenvalues of } A
\end{array} \]

\[ \begin{array}{c}
d_3 - s \quad d_2 - s \quad d_1 - s \\
\text{eigenvalues of } A - sI
\end{array} \]

\[ \begin{array}{c}
\frac{1}{d_3 - s} \quad \frac{1}{d_2 - s} \quad 0 \quad \frac{1}{d_1 - s} \\
\text{eigenvalues of } (A - sI)^{-1}
\end{array} \]

For these eigenvalues/choice of shift $s$, \( \left( \frac{1}{d_2 - s} \right) \) is dominant for \( (A - sI)^{-1} \).

To compute $d_2$:
1. Find $s$ close to $d_2$
2. Powering with \( (A - sI)^{-1} \) to find \( \left( \frac{1}{d_2 - s} \right) \)
3. Solve for $d_2 = s + 1/(1/(d_2 - s))$
daverse powering with shift:
Start with $\hat{x}^0$.

Form $\hat{x}^{(1)} = (A-sI)^{-1}\hat{x}^0$,
$\hat{x}^{(2)} = (A-sI)^{-1}\hat{x}^{(1)}$

\[\vdots\]

\[\text{etc}\]

Suppose find a dominant eigenvalue $\mu$ for $(A-sI)^{-1}$.

\[(A-sI)^{-1}\hat{v} = \mu \hat{v}\]
\[\hat{v} = \mu (A-sI)\hat{v}\]
\[(A-sI)\hat{v} = \frac{1}{\mu} \hat{v}\]

\[A\hat{v} = s\hat{v} + \frac{1}{\mu} \hat{v} = \lambda \hat{v}, \text{ where } \lambda = s + \frac{1}{\mu}.\]
12.2 QR algorithm
Simultaneous iteration: Assume $A$ real symmetric.
Note $v_1, \ldots, v_n$ are not necessarily eigenvectors.
Start with $\{v_1, v_2, \ldots, v_n\}$ such that $v_i \neq \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}$.
that are pairwise orthogonal.
Apply powering to each of the $v_1, \ldots, v_n$
(treat each like $x^{(0)}$ above)

$$\begin{bmatrix} A \left( \begin{array}{c} 0 \\ \vdots \\ 0 \end{array} \right) & A \left( \begin{array}{c} 0 \\ \vdots \\ 0 \end{array} \right) \\
\left( \begin{array}{c} 0 \\ \vdots \\ 0 \end{array} \right) & A \left( \begin{array}{c} 0 \\ \vdots \\ 0 \end{array} \right) \end{bmatrix}$$

then after applying $A$ to each.

these column vectors are not, in general, orthogonal.
$R$ orthogonalize by factoring, as

\[
\begin{bmatrix}
\bar{q}_1' & | & \bar{q}_2' & | & \ldots & | & \bar{q}_n' \\
\bar{r}_1 & | & \bar{r}_2 & | & \ldots & | & \bar{r}_n
\end{bmatrix}
\begin{bmatrix}
r_{11} & r_{12} & \cdots & r_{1n} \\
0 & r_{22} & \cdots & r_{2n} \\
0 & 0 & \ddots & r_{nn}
\end{bmatrix}
\]

\[
\text{or: } A I = \bar{G}_1 R_1, \text{ where } \bar{G}_1 \text{ is orthogonal,} \\
R_1 \text{ is upper triangular.}
\]

Now use the vectors 
\[
\bar{q}_1', \bar{q}_2', \ldots, \bar{q}_n'
\]
as the orthogonal set of vectors 
at the next step in the power iteration modified

Apply $A$ : find

\[
\begin{bmatrix}
A\bar{q}_1' & A\bar{q}_2' & \cdots & A\bar{q}_n'
\end{bmatrix}
\text{not necessarily orthogonal}
so reorthogonalize:

write \[
\begin{bmatrix}
Aq_1' & Aq_2' \\
Aq_3' & \cdots & Aq_n'
\end{bmatrix} = \bar{Q}_2 \bar{R}_2
\]

\[
= \begin{bmatrix}
\bar{q}_1 & \bar{q}_2 & \cdots & \bar{q}_n
\end{bmatrix}
\begin{bmatrix}
\ddots & \ddots & \cdots & \ddots \\
\ddots & \ddots & \cdots & \ddots \\
\ddots & \ddots & \cdots & \ddots \\
\ddots & \ddots & \cdots & \ddots
\end{bmatrix}
\begin{bmatrix}
\ddots & \ddots & \cdots & \ddots \\
\ddots & \ddots & \cdots & \ddots \\
\ddots & \ddots & \cdots & \ddots \\
\ddots & \ddots & \cdots & \ddots
\end{bmatrix}
\]

\[
A \bar{Q}_1 = \bar{Q}_2 \bar{R}_2.
\]

next step: form

\[
A \bar{Q}_2, \text{ then re-orthogonalize}
\]

\[
A \bar{Q}_2 = \bar{Q}_3 \bar{R}_3
\]

et

A matrix form of modified power iteration.
that "searches" for all \(n\) eigenvectors simultaneously.
- Makes sense for \(A\) symmetric.
Claim: Equivalent to:

\[ Q_0 = I, \]
\[ A_0 = A Q_0 = Q_1 R', \quad \text{form } A Q_0 \text{ factor as } Q_1 R', \]
\[ A_1 = R_1 Q_1 = Q_2 R_2', \quad \text{form } R_1 Q_1 \text{ factor as } Q_2 R_2'. \]
\[ A_2 = R_2 Q_2 = Q_3 R_3', \quad \text{form } R_2 Q_2 \text{ factor as } Q_3 R_3'. \]

The text p. 540-541 shows equivalence.

Claim: all of matrices \( A_j \) are similar:

\[ A_{j-1} = Q_j R_j = A Q_j R_j Q_j Q_j^T = Q_j (R_j Q_j) Q_j^T \]
\[ R_j = R_j' \]
\[ \det(A_{j-1} - dI) = \det(Q_j A_j (Q_j^T - dI)) \]
\[ = \det(Q_j (A_j - dI) Q_j^T) = \det(A_j - dI). \]

This is called the unshifted QR algorithm.
13.1 Unconstrained optimization without derivatives
Ch. 13 Optimization

Concentrate on minimization; maximization basically the same.
(Just change the objective function)

The function \( f = f(x) \) we try to minimize, is called the **objective function**.

Optimization problems are classified as unconstrained or constrained:

- **Unconstrained**: minimize \( f(x, y) \) given by (complicated formula)
- **Constrained**: minimize \( f(x, y) \) given by (complicated formula) subject to: \( y < x^2 \) nonlinear constraint
Linear programming deals with optimization of linear functions, subject to linear constraints: special topic.

Generally: unconstrained minimization easier than constrained, linear objective functions easier than nonlinear.

One-dimensional problems easier than two, two easier than three, etc.

Problems with a function to minimize, where derivatives can be found, are easier than problems where derivatives are not available.
Minimization

Minimization of functions $f(x)$ of 1 variable $x$

Define: $f$ is unimodal if:

- There is one point at which $f$ attains its minimum value; say $x = x_0$,
- and: $f$ is increasing for $x > x_0$
- decreasing for $x < x_0$.

An example: $f(x) = |x+1|$.

```
-1  0  1
```

---

This page discusses the concept of unimodal functions and provides an example to illustrate the idea. The diagram and notation are used to visually represent the increasing and decreasing nature of the function for different values of $x$. The example function $f(x) = |x+1|$ is shown with critical points at $x = -1$, $x = 0$, and $x = 1$, highlighting how the function's behavior changes across these points.
"Golden section" and the Golden Section Search.

\[ x_1 = 1 - g \]
\[ x_2 = g \]

Interval scaled by \( g \)

\[ g \text{ has property: } \quad g x_2 = x_1 = 1 - g \]

\[ g^2 = 1 - g \]
\[ g^2 + g - 1 = 0 \]
\[ g = \frac{-1 \pm \sqrt{1 + 4}}{2} \]
\[ g = \frac{-1 + \sqrt{5}}{2} \]

Since \( g > 0 \), choose '+':
\[ g = \frac{-1 + \sqrt{5}}{2} \].
C.S.S.: Suppose $f$ unimodal (min) on $[a, b]$, say $[0, 1]$

![Diagram](image)

from values $f(x_1), f(x_2)$ obtain information:

If: $f(x_2) > f(x_1)$, then it must be that $x_1 < x_2$.

($x_1$ cannot be located in $x_1 \geq x_2$)

If: $f(x_2) < f(x_1)$, it must be that $x_1 > x_2$.

(usual case $f(x_2) < f(x_1)$).
then

\[ x_1 \]

= \[ x_1 \]

now need \( f(x_1') \) (one for evaluation)

(already know \( f(x_2') = f(x_1) \) previous step)

now need \( f(x_2') \) (just one for eval. know)

\( f(x_1') = f(x_2) \) previous step

then either

\[ x' \]

or

\[ x' \]
Successive Parabolic interpolation.

Similar, but:

fit a parabola to \( f \), based on values at 3 points.

Know:
\[
\begin{align*}
y_1 &= f(x_1) \\
y_2 &= f(x_2) \\
y_3 &= f(x_3) \\
\end{align*}
\]

Form \( P(x) = a + bx + cx^2 \) such that
\[
\begin{align*}
P(x_1) &= y_1 \\
P(x_2) &= y_2 \\
P(x_3) &= y_3
\end{align*}
\]

then solve \( P'(x) = 0 \) for \( x_4 \);

evaluate \( f(x_4) = y_4 \).

new \( (x_1', y_1'), \ (x_2', y_2'), \ (x_3', y_3') \) :

discard previous pair \( x_i, y_i \)
when \( y_i = \max(y_1, y_2, y_3, y_4) \).
Related to unconstrained minimization

\[ y = c_1 + c_2 t \]

\[ \begin{array}{c}
   y_1 \\
   y_2 \\
   y_3 \\
\end{array} \]

\[ \begin{array}{c}
   t_1 \\
   t_2 \\
   t_3 \\
\end{array} \]

\[ (t_1, y_1), (t_2, y_2), (t_3, y_3) \] are given. The line \( y = c_1 + c_2 t \) approximates this data.

\[ \text{err}_i = y_i - (c_1 + c_2 t_i), \quad \text{err}_2 = y_2 - (c_1 + c_2 t_2), \quad \text{err}_3 = y_3 - (c_1 + c_2 t_3) \]

To minimize \( \beta(c_1, c_2) = \sum_{i=1}^{3} (\text{err}_i)^2 \), by choosing \( c_1, c_2 \) so \( \beta \) attains its minimum value.

One way to regard this problem:

"solve" overdetermined system in sense of finding a least squares solution:

\[
\begin{pmatrix}
1 & t_1 \\
1 & t_2 \\
1 & t_3 \\
\end{pmatrix}
\begin{pmatrix}
c_1 \\
c_2 \\
\end{pmatrix}
= 
\begin{pmatrix}
y_1 \\
y_2 \\
y_3 \\
\end{pmatrix}
- 
\begin{pmatrix}
\text{err}_1 \\
\text{err}_2 \\
\text{err}_3 \\
\end{pmatrix}
\]
The vector $\hat{c}$ (least squares coefficients) satisfies
\[ M^T M \hat{c} = M^T \hat{y} \]
of form, \[ A \hat{c} = \hat{b} \]
where: \[ A = M^T M \] is symmetric,
and: \[ \hat{x}^T A \hat{x} > 0 \] any \( \hat{x} \neq (0) \).

A is positive definite.
13.1 Unconstrained optimization without derivatives
methods for minimization of a function of 1 variable, may be applied to perform line search in 2 or more dimensions.

\[ f(x, y) = (x-2)^2 + (y-3)^2 \]

\[(x_0, y_0) = (7, 7)\]

Then \[ F(s) = f((x_0, y_0) + s \vec{v}) \]

\[ = f((7, 7) + 5(1, 2)) \]

\[ = f(7 + 5, 7 + 10) \]

is a function of 1 variable \( s \).
13.1.3 Nelder-Mead search
Minimization of functions of $n$ variables:

$$\min_{\vec{x}} f(\vec{x}), \quad \vec{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \in \mathbb{R}^n$$

Nelder-Mead:

Construct a simplex, set of $n+1$ vertices in $\mathbb{R}^n$

$\mathbb{R}^2$ $\quad \mathbb{R}^3$

$\vec{x}_1$ $\quad \vec{x}_3$

$\vec{x}_2, \vec{x}_4$ $\quad$ triangle $\quad$ tetrahedron

$\mathbb{R}^3$

N-M idea: at each step, replace "worst" vertex with some new point, where function value is smaller than at any of the current vertices.
\[ \bar{x} = \text{vertex with largest value of } y = f(x) \]

\[ \bar{x} = \text{average of the current set of vertices except for } x_{\text{n}} \]

Consider the line through points \( x_{\text{n}} \) and \( \bar{x} \). Along this line, consider some special points.

Consider:

\[ x = \bar{x} + 2(\bar{x} - x_{\text{n}}) = 3\bar{x} - 2x_{\text{n}} \]
\[ x = \bar{x} + 1.5(\bar{x} - x_{\text{n}}) = 2.5\bar{x} - 1.5x_{\text{n}} \]
\[ x = \bar{x} + 1(\bar{x} - x_{\text{n}}) = 2\bar{x} - x_{\text{n}} \]
\[ x = \bar{x} + 0.5(\bar{x} - x_{\text{n}}) = 1.5\bar{x} - 0.5x_{\text{n}} \]
\[ x = \bar{x} - 0.5(\bar{x} - x_{\text{n}}) = 0.5\bar{x} + 0.5x_{\text{n}} \]

(At this point, switched to looking at the code maderneadv_plot.py, and the animation)
13.2.1 Newton’s method
Newton's method for \( \min_{x} f(x), \quad x \in \mathbb{R}^n \)

Find \( x^* \) such that \( \frac{\partial f}{\partial x_1} = \frac{\partial f}{\partial x_2} = \ldots = \frac{\partial f}{\partial x_n} = 0 \) at \( x = x^* \).

System of equations:

\[
\begin{bmatrix}
\frac{\partial f}{\partial x_1} \\
\frac{\partial f}{\partial x_2} \\
\vdots \\
\frac{\partial f}{\partial x_n}
\end{bmatrix} \text{ solve } F(x) = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} \text{ for } x^*.
\]

Can apply Newton's method to solve \( F(x) = 0 \).

Initial guess \( x_0 \).

\[
x_1 = x_0 - (D F(x_0))^{-1} F(x_0)
\]

solution of linear alg.

\( \text{e.g., } D F(x_0) s = F(x_0) \).

where \( D F(x_0) = \text{Jacobian of } F, \text{ at } x_0. \)
\[
DF = \begin{bmatrix}
\frac{\partial F_1}{\partial x_1} & \frac{\partial F_1}{\partial x_2} & \cdots & \frac{\partial F_1}{\partial x_n} \\
\vdots & & & \vdots \\
\frac{\partial F_n}{\partial x_1} & \frac{\partial F_n}{\partial x_2} & \cdots & \frac{\partial F_n}{\partial x_n}
\end{bmatrix}, \quad \text{where} \quad F(x) = \begin{bmatrix} F_1 \\ F_2 \\ \vdots \\ F_n \\ \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}
\]

\[
DF = \begin{bmatrix}
\frac{\partial^2 f}{\partial x_1 \partial x_1} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\
\vdots & & & \vdots \\
\frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_n}
\end{bmatrix}
= H(x),
\]

the Hessian matrix

of 2nd partials,

for \( f \).

Since \( \frac{\partial^2 f}{\partial x_i \partial x_j} = \frac{\partial^2 f}{\partial x_j \partial x_i} \), \( H(x) \) is symmetric.

**Newton's method:** given \( x_0 \); for \( k = 1, 2, 3, \ldots \)

compute \( x_k = x_{k-1} - s_{k-1} \), where \( s_{k-1} \) satisfies

\[
\frac{DF(x_{k-1})s_{k-1}}{H(x_{k-1})} = F(x_{k-1}).
\]
13.2.3 Conjugate gradient search
Gradient / Conjugate Gradient methods

to minimize \( f(x, \ldots, x_n) \)

Recall the least squares problem:

Data pairs \((t_i, y_i), i = 0, 1, 2\) would like

\[ c_0 + c_1 t_0 = y_0 \]
\[ c_0 + c_1 t_1 = y_1 \]
\[ c_0 + c_1 t_2 = y_2 \]

But not possible in general

Instead: Given \( \hat{\mathbf{c}} = \begin{pmatrix} c_0 \\ c_1 \end{pmatrix} \), define error \( \hat{\mathbf{\varepsilon}} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \end{pmatrix} - \begin{pmatrix} c_0 + c_1 t_0 \\ c_0 + c_1 t_1 \\ c_0 + c_1 t_2 \end{pmatrix} \)

Let \( \mathbf{M} = \begin{pmatrix} 1 & t_0 \\ 1 & t_1 \\ 1 & t_2 \end{pmatrix} \); then \( \hat{\mathbf{\varepsilon}} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \end{pmatrix} - \mathbf{M} \begin{pmatrix} c_0 \\ c_1 \end{pmatrix} = \hat{\mathbf{y}} - \mathbf{M} \hat{\mathbf{c}} \)

We will choose \( \hat{\mathbf{c}} \) to minimize \( \| \hat{\mathbf{\varepsilon}} \|^2 = \hat{\mathbf{\varepsilon}}^T \hat{\mathbf{\varepsilon}} = (\hat{\mathbf{y}} - \hat{\mathbf{\varepsilon}}^T \mathbf{M}^T \hat{\mathbf{c}})^T \hat{\mathbf{\varepsilon}} \)

Let \( \hat{f}(\hat{\mathbf{c}}) = \| \hat{\mathbf{\varepsilon}} \|^2 = \hat{\mathbf{y}}^T \hat{\mathbf{y}} - \hat{\mathbf{\varepsilon}}^T \mathbf{M} \hat{\mathbf{y}} - \hat{\mathbf{\varepsilon}}^T \mathbf{M} \hat{\mathbf{c}} + \hat{\mathbf{c}}^T \mathbf{M}^T \mathbf{M} \hat{\mathbf{c}} \), symmetric.
Can solve for least-squares $\overline{c} = (c_0 \cdots c_i)$ by minimizing $f(\overline{c})$, which is the same as solving $\frac{df}{dc_0} = 0$ for $c_0, c_i$, $\frac{df}{dc_i} = 0$

"normal equations"

Since $f(\overline{c})$ involves $c_0, c_i, c_0^2, c_i c_0, c_i^2$, these normal equations are linear in $c_0, c_i$, and of form,

$$A \overline{c} = \overline{b}$$

where $A = M^TM$. is symmetric.
A different problem, solvable by minimization. Given $A$, symmetric positive definite, solve $A\hat{x} = \hat{b}$ for $\hat{x}$.

Claim: Can find this $\hat{x}$, by minimizing the objective function

$$f(\hat{x}) = \frac{1}{2} \hat{x}^T A \hat{x} - \hat{x}^T \hat{b}.$$  

(The equations $\frac{df}{d\hat{x}_i} = 0$, reduce to $\sum_{j=1}^{n} A_{ij} x_j - b_i = 0$.)

In Ch. 2, the method called "Conjugate Gradient" solves such systems $A\hat{x} = \hat{b}$. 
Defn: Vectors $\vec{u}$ and $\vec{v}$ are conjugate with respect to $A$, if $\vec{u}^T A \vec{v} = 0$.

(like orthogonality).

Can define an inner product $(\vec{u}, \vec{v})_A = \vec{u}^T A \vec{v}$.

Conjugate Gradient to solve $A\vec{x} = \vec{b}$, terminates in at most $n$ steps; space $\mathbb{R}^n$ is "exhausted" by the collection of search directions $\vec{z}_0, \vec{z}_1, \ldots, \vec{z}_{n-1}$. 
Gradient Search is a method to minimize
\[ f(x_1, x_2, \ldots, x_n) \].

Idea: search in the "downhill" direction.

\[ z = f(x, y) \],
\[ \nabla f = \left( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right) \]
points in direction of most rapid increase of \( f \).

So: \( -\nabla f \) points in direction of most rapid decrease of \( f \).
Given initial guess \( \hat{x}_0 \),

let \( \nabla f(\hat{x}_0) \)

and consider \( f(\hat{x}_0 + s \hat{d}_0) \), \( s \in \mathbb{R} \).

\( \hat{x} = \hat{x}_0 + s \hat{d}_0 \) is the search line.

Use line search: find \( s = \alpha \) that minimizes \( f(\hat{x}_0 + s \hat{d}_0) \).

Let \( \hat{x}_1 = \hat{x}_0 + \alpha \hat{d}_0 \).

Let \( \hat{d}_1 = -\nabla f(\hat{x}_1) \)

and consider \( f(\hat{x}_1 + s \hat{d}_1) \), \( s \in \mathbb{R} \).

find \( s = \alpha \) that minimizes \( f(\hat{x}_1 + s \hat{d}_1) \).
Conjugate Gradient Search is similar to Gradient Search, except the search directions $\hat{d}_1, \hat{d}_2, \ldots$ are chosen to be "conjugate" to each other.

Basic idea: If $f(\hat{x})$ derived from $\frac{1}{2}\hat{x}^TA\hat{x} - \hat{x}^Tb$, then $-\nabla f(\hat{x}) = \hat{\hat{x}}(\hat{x}) = A\hat{x} - b \quad (\text{Eq.})$

C. C. $\hat{x}$:

\[ \hat{x}_0 = \hat{x}_0 \]

find $\alpha = \alpha_0$ that minimizes $f(\hat{x}_0 + \alpha \hat{x}_0)$.

let $\hat{x}_1 = \hat{x}_0 + \alpha \hat{x}_0$; evaluate $\hat{\hat{x}}_1 = \hat{\hat{x}} - A\hat{x}_1$.

find $\beta = \beta_0$ such that $\hat{x}_1 + \beta \hat{x}_0$ is conjugate to $\hat{x}_0$.

let $\hat{x}_1 = \hat{x}_1 + \beta \hat{x}_0$ ( $\hat{x}_1^T A \hat{x}_0 = 0$).
- The formula for $\beta$ can be expressed in terms of the residuals $\hat{r}_0, \hat{r}_1$ without involving the matrix $A$ explicitly.

- The formulas for the residuals can be expressed in terms of the gradients $\nabla f(\hat{x}_0), \nabla f(\hat{x}_1)$ without involving the matrix $A$ explicitly.

  Conjugate Gradient Search

- The resulting algorithm can be applied to functions more general than the quadratic function used to derive the algorithm.

- If C.G.S. is applied to $f(\hat{x}) = \frac{1}{2} \hat{x}^T A \hat{x} - \hat{x}^T b$

  where $A$ is symmetric, positive definite, then it converges in at most $n$ steps.