An Implementation of Adaptive Side Lobe Cancellation in MATLAB®

A. P. Shaw

Electronic Warfare and Radar Division
Defence Science and Technology Organisation

DSTO–TN–0955

ABSTRACT

This report describes an implementation in MATLAB® of an adaptive side lobe canceller system, including a copy of the source code.
Published by

DSTO Defence Science and Technology Organisation
PO Box 1500
Edinburgh, South Australia 5111, Australia

Telephone:   (08) 7389 5555
Facsimile:   (08) 7389 6567

© Commonwealth of Australia 2010
AR No. 014–078
July, 2010
An Implementation of Adaptive Side Lobe Cancellation in MATLAB®

Executive Summary

The objective of an adaptive side lobe cancellation system is to suppress high duty cycle and noise-like interference signals received through the side lobes of the radar. This is accomplished by using auxiliary antennas whose antenna pattern approximates to the side lobe pattern of the main antenna. By suitably phasing the signals received by the auxiliaries a directional anti-phase signal can be generated that when added to the main antenna signal “subtracts” the interference.

This report describes an implementation in MATLAB® of an adaptive side lobe canceller system, including a copy of the source code.
Contents

1 Introduction 1

2 ASLC inputs 1
   2.1 On nomenclature ........................................... 1
   2.2 Sample data .................................................. 1
   2.3 Range strides ............................................... 2
   2.4 Training region ............................................. 2
   2.5 Training data ............................................... 2
   2.6 Pre-computed weights ....................................... 3

3 ASLC outputs 3
   3.1 Output data .................................................. 3
   3.2 Weights data ................................................ 3

4 Hidden parameters 3

5 Limitations 4

References 4

Appendices

A MATLAB implementation of ASLC 5
1 Introduction

The objective of an adaptive side lobe cancellation (ASLC) system is to suppress high
duty cycle and noise-like interference signals received through the side lobes of the radar.
This is accomplished by using auxiliary antennas whose antenna pattern approximates to
the side lobe pattern of the main antenna. By suitably phasing the signals received by the
auxiliaries a directional anti-phase signal can be generated that when added to the main
antenna signal “subtracts” the interference.

A side lobe canceller effectively works by forming a beam from the auxiliary antennas
that points towards the interference source, the signal from this beam is then subtracted
from the signal detected by the main antenna. The problem is how do we determine the
direction in which to point the beam we form from the auxiliary antennas.

Mathematically the problem is to estimate the complex vector of $N$ weights $W = (W_1, W_2, \ldots, W_N)$ to apply to the $N$ signals $V = (V_1, V_2, \ldots, V_N)$ from the auxiliary
antennas in order to minimise up to $N$ directional interference signals in the side lobes of
the main antenna signal ($V_m$). It may be shown [1] that for band limited noise, appropriate
weights may be estimated from:

$$W = \mu M^{-1}R$$  \hspace{1cm} (1)

where

$$M = E\{V'V^T\}$$  \hspace{1cm} (2)

and

$$R = E\{V_mV^*\}$$  \hspace{1cm} (3)

where $\mu$ is an arbitrary scalar $E\{\}$ the expected value, $*$ the complex conjugate, $T$ the
transpose and $^{-1}$ the inverse.

2 ASLC inputs

2.1 On nomenclature

The code nomenclature assumes that the data has been Doppler processed and pulse
compressed, but that is not essential.

The association of particular matrix dimensions with “range” or “Doppler” is one of
convenience (“range” is the second dimension and “Doppler” the third dimension of the
signal and auxiliary data) and need not necessarily align with the user’s data. However, if
there is a change in the association, then it is the responsibility of the user to ensure that
the parameter values provided are appropriate to the interpretation of the data.

2.2 Sample data

The signal data against which ASLC is to be applied is the $Signal$, which consists of
an arbitrary number of “beams” or “channels” in the form of a three dimensional array
such that $(nCs, nRs, nDs)=size(Signal)$ where $nCs$ is the number of signal channels,$nRs$ is the number of range bins and $nDs$ is the number of Doppler bins.
The auxiliary data that is to provide the ASLC is **Auxiliaries**, which consists of an arbitrary number of “beams” or “channels” in the form of a three dimensional array such that \((nCa, nRa, nDa)=\text{size(Auxiliaries)}\) where \(nCa\) is the number of signal channels, \(nRa\) is the number of range bins and \(nDa\) is the number of Doppler bins. It is essential that \(nRa == nRs\) and \(nDa == nDs\).

### 2.3 Range strides

Simple ASLC is implemented with a single set of training data over the data set. This is not desirable if the interference is not uniform over the data set. The baseline code directly supports the segmentation of the data into “range strides”, with calculation of the adaptive weight carried out for each stride. The number of strides used is determined by the \(\text{Nstride}\) parameter. The application of the weights across the data is then by interpolation, so that each range bin has a unique set of weights.

In some circumstances it is desirable to not include data from the range region at the centre of the stride. The size of the excluded region is defined by the \(\text{RangeGap}\) parameter.

### 2.4 Training region

The extent of the training region is established by the \(\text{ClutterExtentIn}\) vector.

If \(\text{length(ClutterExtentIn)}==4\) then the clutter region is assumed to extend over two regions from \(\text{Init}_bin=\text{ClutterExtentIn}(1)\) to \(\text{low}_bin=\text{ClutterExtentIn}(2)\) and from \(\text{high}_bin=\text{ClutterExtentIn}(3)\) to \(\text{final}_bin=\text{ClutterExtentIn}(4)\).

If \(\text{length(ClutterExtentIn)}==2\) then it is assumed that \(\text{Init}_bin=1\) and \(\text{final}_bin=nDa\). The values in \(\text{ClutterExtentIn}\) are sorted into an ascending list so that they cannot overlap, although the regions may merge.

The sample region used is determined by the \(\text{SampleRegion}\) parameter.

When \(\text{SampleRegion}=="\text{noise}"\) then the training is obtained by a single region (from \(\text{low}_bin\) to \(\text{high}_bin\)) in the data and the ASLC corrections are applied over the entire data space.

When \(\text{SampleRegion}=="\text{clutter}"\) then the training is obtained from two sample regions (from \(\text{Init}_bin\) to \(\text{low}_bin\) and from \(\text{high}_bin\) to \(\text{final}_bin\)) and the ASLC corrections are applied only over the training region.

Any other value for \(\text{SampleRegion}\) results in a selection of training data on either side of the middle Doppler bin until the required number of samples has been obtained. The resulting ASLC corrections are applied over the entire data space.

### 2.5 Training data

To train the data, samples from the training region must be selected. The number of samples in range (\(\text{Nranges}\) parameter) and Doppler (\(\text{Ndopplers}\) parameter) to be independently defined.
Radar data may include correlations in range and/or Doppler due to the waveform design, oversampling, compression weights, Doppler padding etc. This can result in the use of correlated data in the training which does not provide useful training data (at least not in the baseline implementation). To overcome this the software supports the ability to require that only every $n^{th}$ data sample is used. The spacing for range space and for Doppler space can differ. If the required number of samples will take the data outside of the region of validity then the software makes appropriate adjustments to the data collected. The spacing between samples is determined by the $N_{\text{step}}$ vector. If $N_{\text{step}}$ is a single value then the value is used for both range and Doppler, if it is an array, then the first value is used for the range dimension and the second value used for the Doppler dimension.

There is no checking that the number of samples is sufficient for an accurate estimate of the covariance matrix.

2.6 Pre-computed weights

In some circumstances it is desirable to use pre-computed weights from a prior data set. This is supported by providing in the output from the function the weights that have been estimated. These weights may then be placed as an input to the function in the variable $\text{InWeights}$. If this function is not required, then this variable must be set to $\text{InWeights}=[]$. The implementation assumes (apart from a few simple checks) that the supplied weights are appropriate to the data set that they are being used on.

3 ASLC outputs

3.1 Output data

The outputs data set is the same format and size as the input signal data, but after ASLC has been applied to the signal.

3.2 Weights data

Weights data that is calculated by the code is supplied as an additional output. This enables examination of the weight magnitudes, but also the use of the weights generated with a similar (same size) signal and auxiliary data (with the other input parameters the same) in accordance with section 2.6.

4 Hidden parameters

The ASLC implementation includes a number of “hard coded” variables that may need to be changed for a particular implementation.
The parameter `ARBITRARY_SCALAR` represents $\mu$ the arbitrary scalar in equation 1 and is currently set to unity.

In training the data there is a simple test (see source code lines 335-339 and 372-373) for the presence of a large target in the training data (a crude cell average CFAR) so that large targets can be excluded from the training data. The magnitude of target that is considered “large” for exclusion is set by the value of `LARGE_TARGET_THRESHOLD`. In the code herein it is set to 20.

If other processing has been applied to eliminate “zero” Doppler effects such as DPCA which has been trained on the same data as will be used for ASLC, then it is (probably) desirable to exclude that data from the ASLC training data. In the code herein, the region that is excluded is determined by the parameter `DPCA_MASKED` and it ensures that Doppler bins $1:DPCA\_MASKED$ and bins $nDa-DPCA\_MASKED:nDa$ are excluded from training.

## 5 Limitations

There is no conditioning applied in the generation of the covariance matrix in this implementation. Depending upon the data and usage of the code it is possible that the estimate for the covariance matrix will be ill-conditioned for inversion. Generally MATLAB® will throw a warning if this is the case.

The MATLAB® code doesn’t make use of the (obvious) $\text{inv}(M) \ast R'$ instead it uses the MATLAB® recommendation of $M \backslash R'$. To change to the explicit code the “commenting out” needs to be changed in the source code at lines 390 and 391.

The code has been modified to MATLAB® 2009b. The majority of the code should work with prior versions except where the dummy variable “~” has been used in some return values. If the code is to be run on an older implementation of MATLAB® then “~” will need to be changed to (any) convenient (non-clashing) variable name in lines 125 and 129.

## References

Appendix A  MATLAB implementation of ASLC

The following is the code used to compute and apply the ASLC weights.

```matlab
function [output, weights]=ASLC_ARCS1(Signal, Auxiliaries, Nranges,...
    Ndopplers, Nstep, Nstride, ...
    SampleRegion, RangeGap, ...
    InWeights, ClutterExtentIn)

%function [output, weights]=ASLC_ARCS1(Signal, Auxiliaries, Nranges,...
%    Ndopplers, Nstep, Nstride, ...
%    SampleRegion, RangeGap, ...
%    InWeights, ClutterExtent)

% Compute and apply the ASLC weights

%Parameters
% Signal: [Signal Range Doppler] array for the signal
% Auxiliaries: [Aux Range Doppler] array for the auxiliaries
% Nranges: Number of range bins to be used in covariance estimate
% Ndopplers: Number of doppler bins to be used in covariance
% Nstep: Number of range & Doppler bins between samples used
% in covariance estimate can be a vector for different
% step sizes [range Doppler] (not used if InWeights=[])
% Nstride: Number of estimates to be made in range space
% SampleRegion: Defines the region within which training data is
% collected relative to the ClutterExtentIn values
% AND the region to which the processing is applied.
% If SampleRegion='noise' then the training data is
% collected in the region low_bin:high_bin (see
% ClutterExtentIn) and is applied to the ENTIRE Doppler
% extent of the data. If SampleRegion='Clutter' then
% training data is collected in two regions
% Ini_bin:Low_bin and high_bin:finalBin (see
% ClutterExtentIn) and the processing is only applied to
% the same region. Other values for SampleRegion define
% a default space in the middle of the Doppler coverage.
% RangeGap: extent of the range gap in around the stride central
% range that is to be used in estimating the weights
% (not used if InWeights=[])
% InWeights: Precomputed weights, if weights are to be estimated
% from the data then this MUST be a null ([[]]) array,
% if it is NOT null, then the value will be used
% (subject to some sanity checks) overwrite Nstride
% and as the ASLC weights. The form of InWeights is
% assumed to match the form of the output parameter
% weights.
% ClutterExtentIn A Either a four value array [Ini_bin low_bin
% high_bin final_bin] that defines the clutter region
% to be [Ini_bin:low_bin] and [high_bin:final_bin],
% or a two value array [low_bin high_bin] that
% assumes Ini_bin=1 and final_bin=nD where nD is the
```
maximum Doppler bin. Note the final list is always
sorted into ascending order.
output: [Signal Range Doppler] array after ASLC processing
weights: [stride Aux] array of the weights.

Notes:
1. Code syntax assumes post-Doppler processing.
2. There is no conditioning of the covariance matrix prior to
   inversion in this code! MATLAB may throw warning messages!

Original by APShaw, 24 July 2009
Modified for clutter processing (RANGE_GAP) by APShaw, 10 Aug 2009
Modified for pre-computed weights in clutter by APShaw, 27 Aug 2009
Modified to exclude very edge Doppler's from 'edge' option
by APShaw, 01 Sep 2009
Modified to speed up processing by eliminating squeeze functions in
the weight computations using a mixture of a squeeze and a reshape.
by APShaw, 08 Sep 2009
Modified to add 'clutter' specific option by APShaw, 08 Sep 2009
Clean up and improve comments to form version for ARCS study
distribution and set the coding conform to MATLAB 2009b. Eliminate
un-used SampleRegion options. Replace fixed offsets with
parameterised values (DPCA_MASKED and LARGE_TARGET_THRESHOLD).
Changed 'ClutterExtent' variable to enable the clutter extent
region to be tricked into other regions or to limited regions of
the clutter. Reformated to suit incorporation into tech note.
by APShaw, 15 Jan 2010

(c) Copyright, Commonwealth of Australia, 2009, 2010

Fixed parameters and early initialisations

arbirtary scalar applied to noise cancellation
ARBITRARY_SCALAR=1;

Set the threshold to be used in determining if there is a target
present in the training data that needs to be excised. Value is
the SNR (linear)
LARGE_TARGET_THRESHOLD=20;

Set the number of bins to be eliminated from the Doppler region due
to the effects of DPCA
DPCA_MASKED=3;

Gap in range data to be used at center of each stride, to prevent
training on "self data".
RANGE_GAP=RangeGap;
Identify the number of signal and the range/Doppler map size

[nS nRs nDs]=size(Signal);

Identify the number of auxiliaries

[nA nRa nDa]=size(Auxiliaries);

Initialise the output array

output=zeros(size(Signal))+1j*zeros(size(Signal));

Validate the input data

determine if we are using pre-computed weights and validate the data

if isempty(InWeights)
  check that the number of weight channels equals the number of aux
  channels
  if ndims(InWeights)==3
    [nWs, nWc]=size(InWeights);
    assert(and(nWs==nS, nWc==nA), 'Number of weight channels must match SIG and AUX channels');
  else
    [nWc]=size(InWeights);
    assert(nWc==nA, 'Number of weight channels MUST match number of AUX channels')
    nWs=1;
    if nS>1
      if there are more signal channels than weights assume
      % applies only to the first channel.
      Signal(2:end,:,:)=[];
    end
    Nstride=nWs;
    COMPUTEWEIGHTS=false;
  else
    COMPUTEWEIGHTS=true;
  end

Check number of signal/auxiliary Doppler s and ranges match

assert( and(nRa==nRs, nDa==nDs),
  'Range and Doppler bin size of signal and auxiliaries MUST match');

Check the ClutterExtent values

if isempty(ClutterExtentIn)
  if length(ClutterExtentIn)==2
    ClutterExtent=[1 ClutterExtentIn(1) ClutterExtendIn(2) nDs];
  elseif length(ClutterExtentIn)>=4
    ClutterExtent=ClutterExtentIn;
  else
disp('Clutter extent badly defined using default region');
SampleRegion='default';
ClutterExtent=[1 nDs nDs nDs];
end
ClutterExtent=sort(ClutterExtent);
else
disp('Clutter extent is not defined using default region');
SampleRegion='default';
ClutterExtent=[1 nDs nDs nDs];
end

% Set the values for the step sizes in range and Doppler depending
% upon the value(s) passed by Nstep.
if length(Nstep)==1
DNranges=max(Nstep,1);
DNdopplers=max(Nstep,1);
else
DNranges=max(Nstep(1),1);
DNdopplers=max(Nstep(2),1);
end
DNranges_half=floor(Nranges.*DNranges./2);
DNdopplers_half=floor(Ndopplers.*DNdopplers./2);

% Check the number of ranges/dopplers is within the size of the arrays
assert( ((Nranges.*DNranges))<= nRa,...
'Value of <Nranges*Nstep> larger than number range bins in data');
assert( Ndopplers.*DNdopplers<= nDa,...
'Value of <Ndopplers*Nstep> larger than doppler bins in data');

% Determine regions to be used for estimations

% compute the center range bin index for each stride
start=DNranges_half+1;
stop=max(nRa-DNranges_half,start);
check=[];
if Nstride>1
strides=floor(start+(stop-start)*(0:Nstride-1)/(Nstride-1));
% remove any strides with the same value (to prevent redundant
% calculations and interpolation failing)
check_count=0;
for count=2:Nstride
if strides(count)==strides(count-1)
check_count=check_count+1;
check(check_count)=count; %#ok<AGROW>
end
end
if check_count
strides(check)=[];
end
else


```matlab
strides=floor(nRa./2);
end
Nstride=length(strides);

% compute the centres of the Doppler samples and generate a list of the
% Doppler bins to be used for weight estimation.
Doppler_step=max(1,DNdopplers);
SampleRegion=lower(SampleRegion);
switch SampleRegion
    case 'clutter'
        % Make a list equal to the entire Doppler space
        Doppler_list0=1:Doppler_step:nDs;
        % Eliminate those bins that are NOT in the clutter region
        Doppler_list0(...
            and(Doppler_list0 > ClutterExtent(2), ...)
        Doppler_list0 < ClutterExtent(3)))=[];
        Doppler_list0(Doppler_list0 < ClutterExtent(1))=[];
        Doppler_list0(Doppler_list0 > ClutterExtent(4))=[];
        % Use the eliminated region as the Doppler list for training
        Doppler_list=Doppler_list0;
        % remove the DPCA_masked region from the training list
        Doppler_list(Doppler_list <=DPCA_MASKED)=[];
        Doppler_list(Doppler_list >=nDs-DPCA_MASKED)=[];
    case 'noise'
        % Make a list equal to the entire Doppler space
        Doppler_list0=1:Doppler_step:nDs;
        % Eliminate those bins that are in the clutter region
        Doppler_list0(Doppler_list0 <= ClutterExtent(2))=[];
        Doppler_list0(Doppler_list0 >= ClutterExtent(3))=[];
        % Use the eliminated region as the Doppler list for training
        Doppler_list=Doppler_list0;
    otherwise
        %'default'
        D0=floor(nDs./2);
        Doppler_list = ...
            max(1,D0-DNdopplers_half); ...
            Doppler_step:min(D0+DNdopplers_half,nDa);
end
NumberDopplers=length(Doppler_list);

% Failsafe provision - ensure we have at least one Doppler in the list
if NumberDopplers<1
    Doppler_list=floor(nDa./2);
    NumberDopplers=1;
end

%-----------------------------------------------------------
% Estimate the weight sets for each stride
%-----------------------------------------------------------
if COMPUTEWEIGHTS
    % initialise the weights and the intermediate matrices
```
weights=zeros(Nstride,nA,nS)+1j.*zeros(Nstride,nA,nS);
M_init=zeros(nA)+1j.*zeros(nA);
R_init=zeros(nS,nA)+1j.*zeros(nS,nA);
total_power=ones(nS,1);

% loop over the number of range strides to take over the data set
for stride_counter=1:Nstride
   % Determine the extent of region where range data is to be
   % taken, making accommodation for the gap in the range data
   % and retaining the specified number of range bins in the
   % data set.
   Range1=strides(stride_counter)-DNranges_half-RANGE_GAP;
   Range2=strides(stride_counter)+DNranges_half+RANGE_GAP;

   % If the range extents go beyond the limits of valid data,
   % adjust them to be within the range limits
   if Range2>nRa
      Excess=Range2-nRa;
      Range1=Range1-Excess;
      Range2=nRa;
   end
   if Range1<1
      Excess=1-Range1;
      Range2=Range2+Excess;
      Range1=1;
   end
   % Double check the effects of the offsets incase a large
   % number of range bins are in use.
   Range1=max(1,Range1);
   Range2=min(Range2,nRa);

   % Determine the extent of the region where there is a gap
   % in the range data.
   Skip1=strides(stride_counter)-RANGE_GAP;
   Skip2=strides(stride_counter)+RANGE_GAP;

   % Check the guard region doesn’t go beyond the limits of valid
   % data and if they do adjust them accordingly
   if Skip2>nRa
      Excess=Skip2-nRa;
      Skip1=Skip1-Excess;
      Skip2=nRa;
   end
   if Skip1<1
      Excess=1-Skip1;
      Skip2=Skip2+Excess;
      Skip1=1;
   end
   % Double check in case a large value of RANGE_GAP is in use.
   Skip1=max(1,Skip1);
   Skip2=min(Skip2,nRa);
% Make a list of the range bins to be used for this stride and
% then remove those that are in the skip region
ValidRanges=Range1:DNranges:Range2;
ValidRanges(and(ValidRanges>=Skip1, ValidRanges<=Skip2))=[];

% Fail-safe provision: ensure that we have at least one range
% bin left in the stride.
if length(ValidRanges)<1
    ValidRanges=strides(stride_counter);
end

% generate a vector of total signal powers in this stride
% accommodating the gap.
for counter=1:nS
    total_power(counter,:) = squeeze(...
        sum(sum(Signal(nS,ValidRanges,Doppler_list)...
        .*conj(Signal(nS,ValidRanges,Doppler_list))));
end
cells=squeeze((length(ValidRanges).*length(Doppler_list))-1);

% initialise weight computation
M=M_init;
R=R_init;

% compute the covariance matrix for this range stride
for range_counter=1:length(ValidRanges)
    range_bin=ValidRanges(range_counter);

    % Speed up functionality: extract data for this range bin
    T_Signal=squeeze(Signal(:,range_bin,:));
    if nS==1
        T_Signal=T_Signal.';
    end
    T_Auxiliaries=squeeze(Auxiliaries(:,range_bin,:));

    % speed up functionality: reshape to avoid "squeeze" inside
    % the inner loop.
    t_signal=reshape(T_Signal,size(T_Signal,1)...
        *size(T_Signal,2),1);
    t_Auxiliaries=reshape(T_Auxiliaries,...
        size(T_Auxiliaries,1)*size(T_Auxiliaries,2),1);

    for doppler_counter=1:NumberDopplers
        doppler_bin=Doppler_list(doppler_counter);
        Svector=t_signal((doppler_bin-1)*nS+1:...
             (doppler_bin-1)*nS+nS);
        Spower=Svector.*conj(Svector);

        % If there isn't a large target present in the test
        % cell then add this cell
        if ~(max((Spower./(total_power-Spower))./cells)...
             >LARGE_TARGET_THRESHOLD))
\[ A_{\text{vector}} = t_{\text{Auxiliaries}}((doppler_{\text{bin}} - 1)_{nA+1} : \ldots \]
\[ (doppler_{\text{bin}} - 1)_{nA+nA}; \]
\[ M \leftarrow M + (A_{\text{vector}}) \cdot A_{\text{vector}}'; \]
\[ R \leftarrow R + (S_{\text{vector}}) \cdot \text{conj}(A_{\text{vector}}).'; \]
\[ \% \quad \text{counter} = \text{counter} + 1; \]
\end{verbatim}

\[ A_{\text{vector}} = t_{\text{Auxiliaries}}((doppler_{\text{bin}} - 1)_{nA+1} : \ldots \]
\[ (doppler_{\text{bin}} - 1)_{nA+nA}; \]
\[ M \leftarrow M + (A_{\text{vector}}) \cdot A_{\text{vector}}'; \]
\[ R \leftarrow R + (S_{\text{vector}}) \cdot \text{conj}(A_{\text{vector}}).'; \]
\[ \% \quad \text{counter} = \text{counter} + 1; \]
\end{verbatim}

\[ \% \text{Normalisations turned off to save computational cycles} \]
\[ M \leftarrow M ./ \text{counter}; \]
\[ R \leftarrow R ./ \text{counter}; \]

\[ \% \text{Compute the weights: note MATLAB recommendation to not} \]
\[ \% \text{use inv function.} \]
\[ \% \text{weights}(\text{stride}_{\text{counter}}, : , :) = \text{inv}(M) \cdot R'; \]
\[ \text{weights}(\text{stride}_{\text{counter}}, : , :) = M \cdot R'; \]

\[ \% \text{need to take complex conjugate of the weights – this is because} \]
\[ \% \text{we have allowed MATLAB to do the conjugate transpose instead of} \]
\[ \% \text{the transpose as this is marginally faster on the test machine.} \]
\[ \text{weights} = \text{conj}(\text{weights}); \]
\[ \text{else} \]
\[ \% \text{If we are not computing the weights from the data, then use the} \]
\[ \% \text{weights that have been supplied in the input parameters.} \]
\[ \text{weights} = \text{InWeights}; \]
\[ \text{end} \]

\[ \% \text{Apply the weighted auxiliaries to the signal} \]

\[ \% \text{Range values, using "start" and "stop" (calculated previously) values}\]
\[ \% \text{to avoid extrapolation} \]
\[ \text{ranges} = \min(\max(1 : nRs, \text{start}), \text{stop}); \]
\[ \% \text{the Doppler expansion of the range weights to simplify the weight}\]
\[ \% \text{application} \]
\[ \text{expander} = \text{ones}(\text{nDs}, 1); \]
\[ \% \text{initialise the correction matrix} \]
\[ \text{correction0} = \text{zeros}(\text{nRs}, \text{nDs}) + 1j \ast \text{zeros}(\text{nRs}, \text{nDs}); \]
\[ \% \text{initialise the output matrix} \]
\[ \text{output} = \text{zeros}(\text{nS}, \text{nRs}, \text{nDs}) + 1j \ast \text{zeros}(\text{nS}, \text{nRs}, \text{nDs}); \]

\[ \% \text{Apply the weights to each signal in turn} \]
\[ \text{for} \ \text{counter0} = 1 : \text{nS} \]
\[ \% \text{extract the weights for the current signal} \]
\[ \text{Sweights} = \text{squeeze}(\text{weights}(::, \text{counter0})); \]
\[ \% \text{initialise the correction for this signal channel} \]
\[ \text{correction} = \text{correction0}; \]
%compute the weight to be applied at each range bin (output weights
%array has dimensions nA x nRs
if Nstride==1
  % one weight applies to every range bin
  Wout=((squeeze(Sweights(1,:))).')*ones(1,nRs);
  % compute the correction factor
  for counter=1:nA
    correction=correction+...
    squeeze(Auxiliaries(counter,:,:)).*...
    ((expander*Wout(counter,:)).');
  end
else
  % Different weights for each range bin
  for counter=1:nA
    % interpolate the computed strides to each range bin for
    % each array for this signal channel
    Win=squeeze(Sweights(:,counter));
    Wout(counter,:)=interp1(strides,Win,ranges,'spline');
    % compute the correction factor
    correction=correction+...
    squeeze(Auxiliaries(counter,:,:)).*...
    ((expander*Wout(counter,:)).');
  end
end
% apply the correction to the signal, including the arbitrary scalar
% adjustment of the weighted correction.
switch SampleRegion
  case 'clutter'
    % Initially make the output equal to the input
    output(counter0,:,:)=squeeze(Signal(counter0,:,:));
    % Apply the correction only over the entire clutter region
    if Nstride==1
      % Note - this does include the DPCA_MASKED region.
      output(counter0,:,Doppler_list0)=...
      squeeze(Signal(counter0,:,Doppler_list0))=...
      correction(:,Doppler_list0)*ARBITRARY_SCALAR;
    otherwise
      output(counter0,:,:)=squeeze(Signal(counter0,:,:))=...
      correction.*ARBITRARY_SCALAR;
  end
end
An Implementation of Adaptive Side Lobe Cancellation in MATLAB®

**Abstract**

This report describes an implementation in MATLAB® of an adaptive side lobe canceller system, including a copy of the source code.