Great question.

If you configured the **To Workspace** block to save the variable as `out`, then `out` will contain your simulation data directly — typically as:

- A **structure** (`Structure`, `Structure With Time`), or
- An **array** (`Array`, `Timeseries`) depending on what you selected in the block settings.

✅ Case 1: If `Save format = Array`

This is the simplest. You can directly use:

```
matlab
CopyEdit
plot(out(:,1));              % Plot the first signal/channel over time
imagesc(out.');              % View the entire matrix as an image
```

🔁 `out` is assumed to be a matrix: `[samples x channels]`

✅ Case 2: If `Save format = Structure With Time`

Then `out` is a structure like:

```
matlab
CopyEdit
out.time         % Time vector
out.signals.values   % Actual signal data
```

You can plot it as:

```
matlab
CopyEdit
plot(out.time, out.signals.values(:,1));   % First channel over time

imagesc(out.signals.values.');             % Full matrix heatmap
xlabel('Time Index');
ylabel('Channel Number');
```

✅ Case 3: If you used logsout (Logged Signals)

Then access with:

```
matlab
CopyEdit
y = out.getElement(1).Values.Data;    % Or use the name:
out.getElement('MySignal')
```

```
plot(y(:,1));
imagesc(y.');
```

✅ Pro Tip: Check the shape

To confirm what you got from simulation, after it ends, run:

```
matlab
CopyEdit
whos out
```

This will show you:

- Type (array, struct, etc.)
- Size (e.g., 20000×121)
- How to index it properly

Let me know your `To Workspace` block settings (`Array`, `Structure`, etc.) and I'll give you the **exact code** for your case.

Thanks for the screenshot — this dialog appears when you're viewing a **multidimensional signal** in a **Simulink Scope** or **Array Plot**, and it's asking how to interpret the data.

✅ Best Option Depends on Your Signal Shape and Goal

💡 Based on your previous context:

You're likely visualizing something like a **radar data cube slice** (e.g., `[samples × channels]` like `[500 x 8]`, or `[20000 x 121]`), and you're trying to **visualize it efficiently**.

✅ Option Breakdown:

🔷 1. Change active subplot to array plot

- 🔄 Switches to an image-like view
- Best for **matrix visualization**, like a **heatmap**

✅ **Recommended** for:

- Radar returns
- STAP filtered output
- Range-Doppler or range-angle visualization

🥈 *2. Convert to channels (4000)*

- Treats each **column as a separate signal**
- Will plot **4000 traces** (overloaded, slow)

⚠ **Not recommended** for large signals
Use **only if you want to plot a few channels**

🥉 *3. Convert frames (1x8, frame size 500)*

- Treats the signal as a **frame-based stream**
- For DSP-style block-by-block processing

Useful only if you are **streaming signals with frames**
❌ **Not relevant for full matrix visualization**

✅ Recommendation for You

Option	Use When…	Verdict
✅ **Array Plot**	You want to see the full matrix (e.g., samples vs channels) like an image	✔ **Best for STAP / radar**
❌ Convert to Channels	You're plotting ≤ 8 signals	**Not usable for 100s or 1000s of channels**
❌ Convert Frames	You're doing streaming or fixed-frame DSP	**Not needed for your case**

✅ So: Select → "Change active subplot to array plot"

This gives you a nice **image (2D heatmap)** of your entire signal matrix — perfect for radar/STAP data cubes.

Let me know if you want to customize the plot scale or labels.

You're now using the **SMI Beamformer block** in Simulink — nice step toward implementing **STAP-like spatial filtering**!

✅ Your Question:

What should be the **Idx input** to the **SMI Beamformer block**?

✅ Understanding the SMI Beamformer Block

The **SMI (Sample Matrix Inversion) Beamformer** processes:

- Input signal `X` of size **[samples × elements]**
- **Steering direction index** via input port `Idx`

It uses `Idx` to pick the **steering direction** from a list you define in the block's mask (under **Scan Angles**).

✅ So What Is `Idx`?

🔹 `Idx` is a **scalar integer** that specifies **which direction** (from the scan angle list) to steer the beam toward.

✅ Example

Suppose in the block's parameters:

- **Scan Angles** = `[-90:1:90]` → total 181 scan angles

Then:

- `Idx = 1` → steer to `-90°`
- `Idx = 91` → steer to `0°`
- `Idx = 121` → steer to `30°`
- `Idx = 181` → steer to `90°`

So, for example, to steer to **30 degrees**, set:

```
Idx = 121;
```

You can provide this via:

- A **Constant block**
- Or a **MATLAB Function block** like:

```
function idx = beamIndex()
%#codegen
idx = 121; % steer to 30 degrees if scan angles are -90:1:90
```

✅ Summary

Parameter	Value
Scan Angles	`-90:1:90`
Idx input	Index to select desired direction (1-based)
Example	`Idx = 121` for 30° azimuth

✅ Optional: Dynamically Select Direction

If you want to **sweep directions** (like for a beam scan):

```
Idx = mod(floor(t * 5), 181) + 1;  % sweep every 0.2 sec
```

Let me know if:

- You want to combine this with **CFAR after beamforming**
- Or simulate **multiple direction scanning** across pulses!

Thanks — that error message tells us exactly what's wrong:

❌ Error:

`X must be real`
Occurs in: **`phased.CFARDetector`** (inside Simulink MATLAB System block)

✅ Meaning:

The **CFAR Detector block** expects the input signal `X` to be a **real-valued vector**, typically:

- A vector of **magnitudes** or **power values**
- E.g., `abs(x)` or `abs(x).^2`

But you're likely feeding it:

- A **complex-valued signal** (from matched filter, beamformer, etc.)

✅ How to Fix It

🔧 Insert an `Abs` Block (or magnitude operation) before the CFAR Detector

Example Fix in Simulink:

```
[Matched Filter / Beamformer Output]
            ↓
         [Abs]            ← Add this!
            ↓
     [CFAR Detector]
```

This will convert the complex signal into its **magnitude**, making it suitable for CFAR.

🧠 Why This Matters:

CFAR compares signal **magnitude or power** to a noise estimate and threshold. Feeding it raw complex data causes this validation error.

✅ Bonus: Power Instead of Magnitude

If your CFAR thresholding is based on **power**, you can use:

- **Math Function block** set to `"square"`
  (i.e., `abs(x).^2`)

Or in a MATLAB Function block:

```
function y = getPower(x)
%#codegen
y = abs(x).^2;
```

✅ Summary

Step	Fix
CFAR expects real input	Use `abs(x)` or `abs(x).^2`
Add before CFAR block	Use `Abs` block (Simulink > Math Ops)
Output is now real-valued	CFAR works correctly ✅

Let me know if you'd like to:

- Use **log-magnitude** (`20*log10(abs(x))`) instead
- Plot detections after CFAR
- Build a multi-range-bin CFAR

Happy to help!

This error occurs because the **Demux block** in Simulink is **not designed for 3D signals** — it only supports **1D or 2D signals**.

● Error Recap:

✕ "Cannot propagate 3-D signal [...] to Demux"

✓ Solutions

🔧 Option 1: Use a MATLAB Function block to extract slices manually

If you're working with a 3D radar data cube:

```
cube: [range × elements × pulses]
```

You can **extract a specific slice** using a **MATLAB Function block**.

✓Example:

```
function slice = extractSlice(cube)
%#codegen
% Extract one pulse (3rd dim) or one range bin (1st dim)
% Example: return 2D slice at pulse 1
slice = cube(:,:,1);  % Result is [range × elements]
end
```

You can then pass the 2D signal to **STAP beamformer**, visualizer, or any further processing.

🔧 Option 2: Reshape the 3D cube into a 2D signal

Sometimes you may want to **flatten** the cube for debugging or passing to generic blocks.

Use a MATLAB Function block like:

```
function flat = flattenCube(cube)
%#codegen
flat = reshape(cube, [], 1);  % Vectorize entire cube
end
```

You can later reshape it back if needed.

✅ What to Use Instead of Demux for 3D Signals

Goal	Use Instead
Extract 2D slice	MATLAB Function block
View cube as image	**Array Plot** or `imagesc(abs(cube(:,:,1)))`
Process cube with STAP	Send full 3D to `STAP SMI Beamformer` directly

⚠ Do NOT Use Demux for 3D Signals

The **Demux block is only for 1D vectors**:

- For example: `[8×1]` → split into 8 outputs

For 3D matrices like radar data cubes, use **MATLAB Function** or **Selector blocks** smartly.

✅ What You Can Do Now

If you're connecting `simulateRadarCubeMF()` to STAP beamformer:

- ✅ Keep output as 3D
- ❌ Don’t pass it through Demux
- ✅ Connect it **directly** to the **STAP SMI Beamformer** block

---

Would you like me to help:

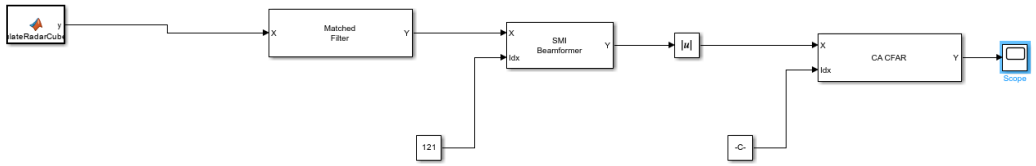
- Add a block to **extract a range bin** for STAP?
- Or extract a pulse for matched filter validation?

I can provide block-level guidance.

CODE RUNNED WELL.

```
function y = simulateRadarCubeMF()
%#codegen
% Returns matched-filtered radar data cube [samples x elements x pulses]

% === Parameters ===
nElements = 8;
nPulses = 32;
fs = 1e6;
prf = 1000;
pw = 1e-4;

% === Waveform ===
waveform = phased.LinearFMWaveform( ...
    'SampleRate', fs, ...
    'PRF', prf, ...
    'PulseWidth', pw);
nSamples = fs / prf;
refPulse = getMatchedFilter(waveform);

% === Antenna and Hardware ===
antenna = phased.ULA('NumElements', nElements);

TX = phased.Transmitter('Gain', 20);
TgtModel = phased.RadarTarget;

tgtPos = [10e3 * sqrt(3); 10e3; 0];
tgtVel = [75 * sqrt(3); 75; 0];
PlatformModel = phased.Platform('InitialPosition', tgtPos, ...
                                'Velocity', tgtVel);
ChannelModel = phased.FreeSpace('TwoWayPropagation', true, ...
                                'SampleRate', fs);

txArray = phased.Radiator('Sensor', antenna, ...
                          'OperatingFrequency', 300e6);
rxArray = phased.Collector('Sensor', antenna, ...
                           'OperatingFrequency', 300e6);
rxPreamp = phased.ReceiverPreamp('Gain', 10, ...
                                 'NoiseFigure', 5, ...
                                 'SampleRate', fs);

radarPos = [0;0;0];
radarVel = [0;0;0];

% === Preallocate ===
dataCube = complex(zeros(nSamples, nElements, nPulses));

% === Loop over pulses ===
for ii = 1:nPulses
    wf = waveform();
    [tgtPos, tgtVel] = PlatformModel(1/prf);
    [~, tgtAng] = rangeangle(tgtPos, radarPos);

    s0 = TX(wf);
    s1 = txArray(s0, tgtAng);
    s2 = ChannelModel(s1, radarPos, tgtPos, radarVel, tgtVel);
    s3 = TgtModel(s2);
    s4 = rxArray(s3, tgtAng);
```

```
    s5 = rxPreamp(s4);

    % === Apply matched filter to each element ===
    for el = 1:nElements
        dataCube(:, el, ii) = conv(s5(:,el), refPulse, 'same');
    end
end

y = dataCube;  % Return complex radar data cube
end
```



Great — your Simulink model uses the output of the `simulateRadarCubeMF()` function (which gives a full **[1000×8×32]** radar data cube) and feeds it to:

1. **Matched Filter** block
2. **SMI Beamformer** block
3. **CFAR Detector**

Let's break down what **`Idx`** should be for the **SMI Beamformer** and **CFAR Detector** blocks.

✓ 1. Index (`Idx`) for the SMI Beamformer

The **SMI Beamformer** expects the full **3D radar data cube**:

```
[dataCube] = [range × channels × pulses]
           = [1000 × 8 × 32]
```

The `Idx` port selects the **range bin** (i.e., target distance) to process using STAP across elements and pulses.

✓ So:

- If you want to apply STAP to **range bin 121**, set:
- `Idx = 121;`
- Range index must be in:
- `1 ≤ Idx ≤ 1000`

This means:
The beamformer will process the slice:

```
rangeBinSlice = squeeze(dataCube(121, :, :));  % [8×32]
```

✓ 2. Index (`Idx`) for the CA CFAR Block

The CFAR Detector takes as input the output of the SMI beamformer:

```
bf_output = [range x 1]
```

The `Idx` for CFAR selects a **specific cell to test**, for example:

- If you set `Idx = 1`, you're testing the **first range cell**
- Typical values: 1 to number of range bins (e.g., 1 to 1000)

However — if your beamformer is producing a **single vector** (over one selected range bin), then CFAR should operate on **that full 1D vector**, not a single index.

✓ Suggested:

- Leave `Idx` unconnected if your CFAR block is configured to automatically slide through the input
- Or use `Idx = 1` if you're manually selecting a test cell

✓ Visual Summary

Block	Input Shape	Idx Meaning	Typical Value
SMI Beamformer	`[1000 × 8 × 32]`	Range bin index	`Idx = 121` (target distance)
CA CFAR	`[1000 × 1]`	Range index to test	`Idx = 1` (or sweep if unconnected)

✅ Pro Tip for Simulink Use

If you're unsure of target range:

- Sweep `Idx = 1:1000` programmatically (in script or UI slider)
- Or manually select based on where target appears strongest (e.g., plot `abs(dataCube(:,1,1))`)

Let me know if you want:

- Help **automating detection** without manual index input
- Plotting the STAP output in Simulink
- Using multiple target scenarios

Happy to support you!

Yes, the **CFAR Detector block parameters** you've shown are **correct and functional**, but here's a breakdown to help you understand and possibly **optimize them** for your radar setup:

✅ Breakdown of Your Settings:

Parameter	Value	Comment
CFAR algorithm	`CA`	Cell-Averaging CFAR is the standard and works well in homogeneous noise.
Guard cells	`2`	Prevents contamination by target signal. 2 is good for narrow targets.
Training cells	`2`	Low. You may want more (e.g., 8–16) for better noise estimation.
Threshold factor method	`Auto`	Fine for most use cases. Can also use `Custom` with empirical scaling.
Probability of false alarm	`0.1`	High. This means 10% false alarm rate. For radar, **0.001 to 0.01** is typical.

Parameter	Value	Comment
Output format	`CUT result`	This gives binary (0/1) detections at each Cell Under Test (CUT).
Simulate using	`Interpreted`	Okay for prototyping. For speed, use **Code generation** if no errors occur.

🔍 Suggested Improvements:

Suggestion	Why
Increase training cells to 8 or 12	More training cells = better noise estimate (especially if cluttered environment).
Reduce `Pfa` to 0.01 or 0.001	More realistic radar false alarm setting.
(Optional) Enable `Output detection threshold`	Useful for plotting threshold vs. signal.
Use `GOCA` or `SOCA` in cluttered environments	More robust than plain `CA` in presence of strong interferers/clutter.

✅ Recommended Configuration (For STAP or Airborne Radar):

Parameter	Recommended
CFAR algorithm	`GOCA` or `SOCA`
Number of guard cells	`2`
Number of training cells	`8–16`
Threshold method	`Auto` or `Custom`
Pfa	`0.001`
Output format	`CUT result`

✅ Visualization Tip:

Connect CFAR output to:

- A **Scope** to see detections over time
- Or a **Product block** to multiply CFAR mask × signal (for plotting detected pulses)

Would you like me to:

- Suggest a MATLAB script for CFAR parameter sweep?
- Generate example detection plots from your cube?

Let me know how deep you'd like to go.

To **visualize a radar data cube** in MATLAB or Simulink, you need to understand its structure and then slice it appropriately. A radar data cube is typically a **3D matrix** of size:

```
[nSamples x nElements x nPulses]
 = (range bins × array elements × pulses)
```

✓ 1. In MATLAB: Visualize from Workspace

If your radar data cube is stored in variable `dataCube`, use these techniques:

---

a. View a Range Profile (1 Pulse, 1 Element)

```
plot(abs(dataCube(:, 1, 1)));
xlabel('Range Bin'); ylabel('Magnitude');
title('Range Profile (Element 1, Pulse 1)');
```

---

b. View Across Array (Angle Profile at fixed range/pulse)

```
rangeBin = 500;
pulseIdx = 1;
plot(abs(dataCube(rangeBin, :, pulseIdx)));
xlabel('Element Index'); ylabel('Magnitude');
title('Spatial Snapshot at One Range Bin');
```

---

c. View Doppler Profile (1 Range, 1 Element)

```
rangeBin = 500;
elementIdx = 1;
doppler = squeeze(dataCube(rangeBin, elementIdx, :));
plot(abs(doppler));
xlabel('Pulse Index'); ylabel('Magnitude');
title('Doppler Profile (Fixed Range & Element)');
```

---

d. Show Range-Doppler Map for 1 Element

```
elementIdx = 1;
rdMap = squeeze(abs(dataCube(:, elementIdx, :))); % size: [range × pulses]
imagesc(rdMap);
xlabel('Pulse Index'); ylabel('Range Bin');
title('Range-Doppler Map for 1 Element');
colorbar;
```

---

e. Volume Visualization (Advanced)

If you want to view the whole cube:

```
volshow(abs(dataCube));  % Needs Image Processing Toolbox
```

✅ 2. In Simulink:

Since the radar data cube is 3D, you **can't directly plot it in Scope**. Instead, use these:

🔧 *Tools to help:*

- **To Workspace block** → set format to `Array` or `Structure With Time`
- **MATLAB Function block** → slice the cube inside it (e.g., select range or pulse slice)
- **Selector block** → to extract specific slice (e.g., 1st element)

Then, plot the extracted 1D or 2D slice using a **Scope block** or **`logsout.plot()`**.

✅ Example (MATLAB Function block to extract one slice):

```
function y = extractRangeProfile(cube)
% Assumes cube is 1000x8x32
y = abs(cube(:,1,1));  % Range profile for element 1, pulse 1
end
```

Would You Like Help With:

- A **Simulink subsystem** to visualize slices?
- Generating **range-Doppler** or **angle-Doppler** maps from Simulink output?

Let me know how you want to explore the data cube visually!