MATLAB 学习笔记

基础语法

1. 变量与数据类型

代码示例与解析

```
% ------ 数值类型 ------
a = 3.14;              % 双精度浮点数（默认）
b = int8(127);         % 8位有符号整数
c = 1 + 2i;            % 复数类型
```

```
% ------ 字符串 ------
str1 = 'Hello';      % 字符数组
str2 = "World";      % 字符串类型（MATLAB R2016b+）

% ------ 结构体 ------
person.name = 'Alice';
person.age = 30;
disp(person.name);   % 输出：Alice

% ------ 单元数组 ------
cell_array = {1, 'text', [2 3]};   % 可混合存储不同类型
disp(cell_array{2});               % 输出：text
```

---

2. 矩阵与数组操作

核心操作示例

```
% ------ 基础矩阵 ------
A = [1 2 3;          % 2x3 矩阵
     4 5 6];
B = zeros(3,2);      % 3x2 全零矩阵
C = rand(2,4);       % 2x4 随机矩阵（0~1均匀分布）

% ------ 索引与切片 ------
val = A(2,3);        % 获取第二行第三列 → 6
col = A(:,1);        % 获取第一列 → [1;4]
submat = A(1:2,2:3);% 子矩阵 → [2 3;5 6]

% ------ 矩阵运算 ------
D = A * B';          % 矩阵乘法（维度需匹配）
E = A .* C;          % 逐元素相乘（维度必须相同）
F = sum(A, 'all');   % 所有元素求和 → 21

% ------ 特殊矩阵 ------
eye_mat = eye(3);    % 3x3 单位矩阵
diag_mat = diag([1 2 3]);   % 对角矩阵
```

MATLAB函数

一、核心基础函数

1. 变量与数据类型

函数	功能描述	示例
whos	查看工作区变量信息	whos
class	查看变量数据类型	class(x)
double	转换为双精度浮点数	x = double(uint8(5))
struct	创建结构体	s = struct('a',1)
cell	创建单元数组	c = {1, 'text'}
table	创建表格数据	T = table(data)

2. 矩阵操作

函数	功能描述	示例
zeros	创建全零矩阵	A = zeros(3,2)
ones	创建全1矩阵	B = ones(5)
eye	创建单位矩阵	I = eye(4)
diag	创建对角矩阵	D = diag([1 2 3])
reshape	矩阵维度重塑	B = reshape(A,2,3)
inv	矩阵求逆	A_inv = inv(A)
det	矩阵行列式计算	d = det(A)

3. 文件与数据IO

函数	功能描述	示例
load/save	加载/保存MAT文件	save('data.mat','x')
readtable	读取表格数据（CSV/Excel）	T = readtable('data.csv')
fopen	打开文件	fid = fopen('data.bin','r')
audioread	读取音频文件	[y,fs] = audioread('test.wav')
imread	读取图像文件	img = imread('test.png')

二、数学与信号处理

1. 基础数学

函数	功能描述	示例
`sin`/`cos`	三角函数	`y = sin(2*pi*f*t)`
`exp`	指数函数	`y = exp(-a*t)`
`fft`/`ifft`	快速傅里叶变换/逆变换	`Y = fft(x)`
`conv`	卷积计算	`z = conv(x,y)`
`filter`	数字滤波	`y = filter(b,a,x)`

2. 信号处理工具箱

函数	功能描述	应用场景
`resample`	信号重采样	调整采样率
`spectrogram`	时频分析（STFT）	非平稳信号分析
`xcorr`	计算互相关/自相关	信号匹配检测
`pwelch`	功率谱密度估计	噪声分析
`fir1`/`fir2`	FIR滤波器设计	数字滤波器设计
`butter`/`cheby1`	IIR滤波器设计	低通/高通滤波器实现
`hilbert`	希尔伯特变换	解析信号提取

3. 雷达信号处理专用

函数	功能描述	工具箱
`phased.LinearFMWaveform`	生成LFM信号	Phased Array Toolbox
`phased.RangeDopplerResponse`	距离-多普勒成像	Phased Array Toolbox
`sar.ImageFormatter`	SAR图像格式化	Radar Toolbox
`stretchproc`	脉冲压缩（去斜处理）	Radar Toolbox

函数	功能描述	工具箱
`cfar`	CFAR检测器	Phased Array Toolbox

三、雷达与通信系统

1. 雷达处理核心

函数	功能描述
`rangeangle`	计算目标距离与角度
`dop2speed`	多普勒频率转速度
`beamscan`	波束形成与扫描
`monopulse`	单脉冲测角处理
`stap`	空时自适应处理（STAP）

2. 通信系统

函数	功能描述
`qammod`/`qamdemod`	QAM调制解调
`pskmod`/`pskdemod`	PSK调制解调
`comm.ErrorRate`	误码率计算
`comm.RayleighChannel`	瑞利信道仿真

四、数据可视化

函数	功能描述	示例
`plot`	二维线图	`plot(x,y)`
`scatter`	散点图	`scatter(x,y)`
`imagesc`	矩阵图像显示	`imagesc(matrix)`

函数	功能描述	示例
surf	三维曲面图	surf(X,Y,Z)
spectrogram	时频图	spectrogram(signal)
stem	离散序列图	stem(x)

五、性能优化与调试

函数	功能描述
tic/toc	代码计时
profile	性能分析
gpuArray	GPU加速计算
parfor	并行循环
memmapfile	内存映射文件处理
dbstop	设置调试断点

六、实用工具函数

函数	功能描述
dir	列出目录内容
which	查找函数路径
ver	查看工具箱版本
input	用户输入对话框
ginput	图形界面坐标拾取

七、完整函数查询方法

1. **按类别查询**:

```
>> help signal            % 查看信号处理函数列表
>> help images            % 查看图像处理函数列表
```

2. **按名称搜索**:

```
>> doc fft                % 打开FFT的官方文档
>> lookfor convolution    % 搜索包含"convolution"的函数
```

3. **工具箱文档**:

- Phased Array Toolbox: `>> doc phased`
- Radar Toolbox: `>> doc radar`
- DSP System Toolbox: `>> doc dsp`

信号处理核心模块

1. 信号生成与可视化

完整代码示例

```
% ------ 参数设置 ------
fs = 1000;                  % 采样率 (Hz)
t = 0:1/fs:1;               % 时间向量(1秒时长)

% ------ 合成信号 ------
f1 = 50;                    % 基频分量
f2 = 150;                   % 高频噪声
signal = 0.7*sin(2*pi*f1*t) + 0.3*cos(2*pi*f2*t);

% ------ 添加噪声 ------
noise_power = 0.2;
noisy_signal = signal + noise_power*randn(size(t));

% ------ 可视化 ------
subplot(2,1,1);
plot(t, signal);
title('原始信号');
xlabel('时间 (s)');
ylabel('幅值');

subplot(2,1,2);
```

```
plot(t, noisy_signal);
title('含噪信号');
xlabel('时间 (s)');
ylabel('幅值');
```

---

2. FFT频谱分析

分步解析与代码

```
% ------ 计算频谱 ------
N = length(noisy_signal);     % 信号长度
Y = fft(noisy_signal);        % 快速傅里叶变换
P2 = abs(Y/N);                % 双边幅度谱
P1 = P2(1:N/2+1);             % 单边频谱
P1(2:end-1) = 2*P1(2:end-1); % 能量校正

% ------ 构建频率轴 ------
f = fs*(0:(N/2))/N;           % 频率坐标(0~fs/2)

% ------ 频谱可视化 ------
figure;
plot(f, P1);
title('单边幅度谱');
xlabel('频率 (Hz)');
ylabel('幅值');
grid on;

% 标注峰值频率
[~, idx] = max(P1);
hold on;
plot(f(idx), P1(idx), 'ro');
text(f(idx)+10, P1(idx), sprintf('%.1f Hz', f(idx)));
```

---

3. 滤波器设计实战

FIR滤波器设计（窗函数法）

```
% ------ 参数设置 ------
order = 60;                   % 滤波器阶数
cutoff_freq = 100;            % 截止频率 (Hz)
nyquist = fs/2;               % 奈奎斯特频率

% ------ 归一化频率 ------
```

```
normalized_cutoff = cutoff_freq / nyquist;

% ------ 滤波器系数生成 ------
b = fir1(order, normalized_cutoff, 'low', hamming(order+1));

% ------ 频率响应可视化 ------
freqz(b, 1, 1024, fs);      % 绘制幅频/相频特性
title('FIR低通滤波器响应');

% ------ 信号滤波 ------
filtered_signal = filtfilt(b, 1, noisy_signal);  % 零相位滤波
```

IIR滤波器设计（Butterworth）

```
% ------ 4阶低通滤波器 ------
[b, a] = butter(4, normalized_cutoff, 'low');

% ------ 零极点分析 ------
zplane(b, a);                % 显示零极点分布
title('Butterworth滤波器零极点图');

% ------ 滤波应用 ------
filtered_signal_iir = filtfilt(b, a, noisy_signal);
```

---

4. 信号分析工具箱

高级分析方法

```
% ------ 短时傅里叶变换 (STFT) ------
window = hamming(256);
noverlap = 200;
nfft = 512;

spectrogram(noisy_signal, window, noverlap, nfft, fs, 'yaxis');
title('时频分析 - 语谱图');

% ------ 自相关分析 ------
[acf, lags] = xcorr(signal, 'coeff');
figure;
plot(lags/fs, acf);
title('信号自相关函数');
xlabel('延迟 (s)');

% ------ 功率谱密度估计 ------
pwelch(noisy_signal, hamming(512), [], [], fs);
title('Welch法功率谱估计');
```

信号处理进阶模块

1. 信号重采样与抗混叠

代码示例与原理说明

```
% ------ 原始信号参数 ------
fs_orig = 500;                % 原采样率 500Hz
t_orig = 0:1/fs_orig:1;       % 原始时间向量
f_signal = 50;                % 信号频率 50Hz
x_orig = sin(2*pi*f_signal*t_orig);

% ------ 重采样至 200Hz ------
fs_new = 200;                 % 新采样率 200Hz
x_resampled = resample(x_orig, fs_new, fs_orig); % 自动抗混叠滤波

% ------ 验证重采样效果 ------
t_new = (0:length(x_resampled)-1)/fs_new;

figure;
subplot(2,1,1);
plot(t_orig, x_orig);
title(['原始信号 (', num2str(fs_orig), 'Hz)']);

subplot(2,1,2);
plot(t_new, x_resampled);
title(['重采样信号 (', num2str(fs_new), 'Hz)']);
xlabel('时间 (s)');

% 原理说明：
% 1. resample 函数会自动应用抗混叠滤波器
% 2. 新采样率需满足 Nyquist 定理（必须大于信号最高频率的2倍）
% 3. 可通过第4个参数控制滤波器阶数：resample(x, p, q, n)
```

2. 数字滤波器设计进阶

多种滤波器类型对比

```
% ------ 设计参数 ------
fs = 1000;                    % 采样率
fc = 100;                     % 截止频率
```

```
order = 6;                    % 滤波器阶数

% ------ Butterworth滤波器 ------
[b_butter, a_butter] = butter(order, fc/(fs/2), 'low');
freqz(b_butter, a_butter, 1024, fs);
title('Butterworth 低通滤波器');

% ------ Chebyshev I型滤波器 ------
[b_cheby1, a_cheby1] = cheby1(order, 0.5, fc/(fs/2), 'low');
figure;
freqz(b_cheby1, a_cheby1, 1024, fs);
title('Chebyshev I型（0.5dB纹波）');

% ------ 椭圆滤波器 ------
[b_ellip, a_ellip] = ellip(order, 0.5, 40, fc/(fs/2), 'low');
figure;
freqz(b_ellip, a_ellip, 1024, fs);
title('椭圆滤波器（0.5dB通带纹波，40dB阻带衰减）');

% 关键区别：
% - Butterworth：最大平坦幅频响应，过渡带较宽
% - Chebyshev：通带等波纹，过渡带比Butterworth窄
% - 椭圆：通带和阻带都有波纹，过渡带最窄
```

---

3. 多信号处理实战：混合信号分离

完整处理流程

```
% ------ 生成混合信号 ------
fs = 2000;
t = 0:1/fs:1;
f1 = 50; f2 = 400; f3 = 800;  % 三个频率分量

signal = 0.5*sin(2*pi*f1*t) + 1.2*cos(2*pi*f2*t) + 0.8*sawtooth(2*pi*f3*t);

% ------ 频谱分析 ------
Y = fft(signal);
L = length(signal);
P2 = abs(Y/L);
P1 = P2(1:L/2+1);
P1(2:end-1) = 2*P1(2:end-1);
f = fs*(0:(L/2))/L;

figure;
plot(f, P1);
title('混合信号频谱');
xlabel('频率 (Hz)');

% ------ 设计带通滤波器提取400Hz分量 ------
[b, a] = butter(4, [380 420]/(fs/2), 'bandpass');
```

```
filtered_signal = filtfilt(b, a, signal);

% ------ 结果可视化 ------
figure;
subplot(2,1,1);
plot(t, signal);
title('原始混合信号');

subplot(2,1,2);
plot(t, filtered_signal);
title('提取的400Hz分量信号');
xlabel('时间 (s)');

% 扩展练习：
% 1. 尝试分离其他频率分量
% 2. 修改滤波器参数观察信号变化
% 3. 添加噪声后测试滤波器性能
```

---

4. 信号处理实用技巧

实战经验总结

技巧1：处理实时信号

```
% ------ 实时信号缓冲处理 ------
buffer_size = 1024;           % 缓冲区大小
data_buffer = zeros(buffer_size, 1);

while true
    new_data = acquire_data();  % 假设的数据获取函数
    data_buffer = [data_buffer(2:end); new_data];  % 更新缓冲区

    % 实时处理（示例：实时滤波）
    processed_data = filter(b_butter, a_butter, data_buffer);

    % 显示更新
    plot(processed_data);
    drawnow;
end
```

技巧2：处理音频文件

```
% ------ 读取音频文件 ------
[y, fs] = audioread('audio_sample.wav');

% ------ 立体声转单声道 ------
if size(y,2) == 2
```

```
    y_mono = mean(y, 2);
else
    y_mono = y;
end

% ------ 降噪处理 ------
% 设计一个带通滤波器保留人声频率范围
[b_voice, a_voice] = butter(4, [80 3000]/(fs/2), 'bandpass');
clean_audio = filtfilt(b_voice, a_voice, y_mono);

% ------ 播放处理结果 ------
sound(clean_audio, fs);
```

---

5. 高级分析方法

小波变换应用

```
% ------ 小波去噪示例 ------
load noisysignal;               % MATLAB 自带示例数据

% 执行小波分解
[c, l] = wavedec(noisysignal, 5, 'db4');  % 5层分解，db4小波

% 阈值处理
thr = wthrmngr('sqtwolog', c, l);          % 自动计算阈值
c_den = wthresh(c, 's', thr);               % 软阈值处理

% 重构信号
denoised_signal = waverec(c_den, l, 'db4');

% 结果可视化
figure;
subplot(2,1,1);
plot(noisysignal);
title('含噪信号');

subplot(2,1,2);
plot(denoised_signal);
title('小波去噪结果');
```

MATLAB雷达信号处理与SAR成像专题指南

一、雷达信号处理核心模块

1. 线性调频信号（LFM）生成与脉冲压缩

代码实现

```
%% LFM信号参数设置
B = 100e6;                % 带宽100MHz
T = 10e-6;                % 脉冲宽度10μs
fs = 2*B;                 % 采样率200MHz
t = -T/2:1/fs:T/2;        % 时间向量
f0 = 10e9;                % 载频10GHz

%% 生成LFM信号
chirp_signal = exp(1j*pi*(B/T)*t.^2) .* exp(1j*2*pi*f0*t);

%% 脉冲压缩处理（匹配滤波）
matched_filter = conj(fliplr(chirp_signal)); % 匹配滤波器
compressed_signal = fftconv(chirp_signal, matched_filter, 'same');

%% 结果可视化
figure;
subplot(211); plot(t*1e6, real(chirp_signal));
title('LFM时域波形'); xlabel('时间(μs)');

subplot(212); plot(abs(compressed_signal));
title('脉冲压缩结果'); xlabel('采样点');
```

关键技术说明

- **带宽-时间积（BT积）**：决定压缩比，本例中BT=100e6×10e-6=1000
- **距离分辨率**：ΔR = c/(2B) = 3e8/(2×100e6) = 1.5m
- **副瓣抑制**：可通过加窗（如Hamming窗）改善，但会加宽主瓣

---

2. 多普勒处理与CFAR检测

多普勒FFT分析

```
%% 多普勒参数设置
PRF = 1000;               % 脉冲重复频率1kHz
num_pulses = 128;         % 相参处理间隔(CPI)脉冲数

%% 模拟含多普勒的目标回波（速度为v的目标）
v = 100;                  % 目标径向速度m/s
lambda = 3e8/f0;          % 波长计算
doppler_freq = 2*v/lambda;
phase_shift = exp(1j*2*pi*doppler_freq*(0:num_pulses-1)/PRF);

%% 多普勒FFT处理
```

```
doppler_fft = fftshift(fft(phase_shift));

%% 可视化
f_axis = (-num_pulses/2:num_pulses/2-1)*PRF/num_pulses;
figure;
plot(f_axis, abs(doppler_fft));
title('多普勒频谱'); xlabel('频率(Hz)'); grid on;
```

CFAR检测实现

```
%% 创建CFAR检测器
cfar = phased.CFARDetector('Method', 'OS', 'Rank', 10,...
                           'NumTrainingCells', 20, 'NumGuardCells', 2);

%% 生成模拟距离单元数据
num_cells = 1000;
noise_power = 1;
signal_power = 100;
data = sqrt(noise_power/2)*(randn(num_cells,1)+1j*randn(num_cells,1));
data(300:310) = sqrt(signal_power/2)*(randn(11,1)+1j*randn(11,1)); % 注入目标

%% 执行检测
detections = cfar(abs(data).^2, 1:num_cells);

%% 结果可视化
figure;
plot(abs(data)); hold on;
plot(find(detections), abs(data(detections)), 'rx');
title('CFAR检测结果'); legend('信号能量','检测目标');
```

二、SAR成像处理流程

1. 距离多普勒算法（RDA）实现

完整成像代码框架

```
%% SAR系统参数
c = 3e8;                    % 光速
fc = 5.3e9;                 % 载频5.3GHz
B = 100e6;                  % 带宽
V = 150;                    % 平台速度m/s
H = 5000;                   % 平台高度m
R0 = sqrt(H^2 + (V*10)^2); % 斜距（假设10秒后位置）

%% 回波数据模拟（以点目标为例）
num_range = 1024;           % 距离向采样点数
```

```
num_azimuth = 512;        % 方位向采样点数
raw_data = zeros(num_azimuth, num_range);
raw_data(:, 400:600) = exp(1j*2*pi*rand(num_azimuth, 201)); % 模拟散射点

%% RDA处理流程
% 距离压缩
range_compressed = fft(raw_data, [], 2);
matched_filter_r = exp(1j*pi*B/T*(t.^2)); % 距离向匹配滤波器
range_compressed = ifft(range_compressed .* matched_filter_r, [], 2);

% 方位向FFT
azimuth_fft = fft(range_compressed, [], 1);

% 距离徙动校正(RCMC)
% 此处需实现插值操作(简化为理想校正)
rcmc_corrected = azimuth_fft;

% 方位压缩
matched_filter_a = exp(1j*4*pi*R0/lambda*(...)); % 方位向匹配滤波器
azimuth_compressed = ifft(rcmc_corrected .* matched_filter_a, [], 1);

%% 成像结果显示
figure;
imagesc(20*log10(abs(azimuth_compressed)));
title('SAR成像结果'); colormap('jet'); colorbar;
```

三、AD采样数据SFDR分析

1. SFDR测量完整流程

```
%% 生成测试信号
fs = 1e9;                 % 采样率1GHz
f_in = 100e6;             % 输入信号100MHz
N = 2^18;                 % 样本点数
t = (0:N-1)/fs;
adc_input = 0.5*sin(2*pi*f_in*t) + 0.001*randn(size(t));

%% 加窗处理
window = hann(N)';
windowed_signal = adc_input .* window;

%% FFT分析
spectrum = fft(windowed_signal)/N;
PdB = 20*log10(abs(spectrum(1:N/2)));
f_axis = (0:N/2-1)*fs/N;

%% SFDR计算
[fund_power, fund_idx] = max(PdB);
spurs = PdB([1:fund_idx-200, fund_idx+200:end]);
[spur_power, spur_idx] = max(spurs);
```

```
SFDR = fund_power - spur_power;

%% 图形化显示
figure;
semilogx(f_axis, PdB); hold on;
plot(f_axis(fund_idx), fund_power, 'ro');
plot(f_axis(spur_idx), spur_power, 'rx');
title(['SFDR = ', num2str(SFDR, '%.1f'), ' dB']);
xlabel('频率 (Hz)'); ylabel('幅度 (dB)');
grid on; xlim([1e6 500e6]);
```

四、数字校准关键技术

1. I/Q失配校准

```
%% 生成I/Q不平衡信号
fs = 100e6;
t = 0:1/fs:1e-6;
f0 = 10e6;

% 引入失配参数
gain_imbalance = 1.2;      % I/Q幅度失配
phase_error = 5;           % 相位误差（度）

% 生成失配信号
I = cos(2*pi*f0*t);
Q = gain_imbalance * sin(2*pi*f0*t + deg2rad(phase_error));

%% 失配估计与补偿
% 计算失配参数
alpha = real(I*Q')/(real(I*I'));     % 正交误差参数
phi = asin(imag(I*Q')/(norm(I)*norm(Q))); % 相位误差

% 补偿矩阵
comp_matrix = [1, -alpha*sin(phi);
               0, 1/cos(phi)];

% 应用补偿
compensated = comp_matrix * [I; Q];

%% 结果验证
figure;
subplot(211); plot(real(compensated), imag(compensated));
title('补偿前I/Q轨迹'); axis equal;

subplot(212); plot(real(compensated), imag(compensated));
title('补偿后I/Q轨迹'); axis equal;
```

五、高级应用技巧

1. 大规模雷达数据处理优化

```
% 使用内存映射处理大文件
memmap_data = memmapfile('radar_data.bin', ...
    'Format', {'single', [1e6 1024], 'data'});

% 分块处理（示例：脉冲压缩）
block_size = 1000;
num_blocks = ceil(size(memmap_data.Data.data, 1)/block_size);

parfor blk = 1:num_blocks
    start_idx = (blk-1)*block_size + 1;
    end_idx = min(blk*block_size, size(memmap_data.Data.data,1));

    block_data = memmap_data.Data.data(start_idx:end_idx, :);

    % 执行脉冲压缩
    compressed_block = fftconv(block_data, matched_filter, 'same');

    % 保存结果
    writeToFile(compressed_block, blk);
end

% 启用GPU加速（需支持CUDA）
if gpuDeviceCount > 0
    gpu_matched_filter = gpuArray(matched_filter);
    gpu_data = gpuArray(block_data);
    compressed_gpu = gather(fftconv(gpu_data, gpu_matched_filter));
end
```

关键工具包推荐：

1. **Phased Array System Toolbox**
   - 提供 `phased.LinearFMWaveform`生成LFM信号
   - 内置 `phased.RangeDopplerResponse`用于距离多普勒显示
2. **Radar Toolbox**
   - 支持SAR成像处理链快速搭建
   - 包含 `sar.ImageFormatter`等专用成像函数
3. **DSP System Toolbox**
   - 提供 `dsp.DynamicFilter`用于实时校准
   - 包含高级滤波器设计工具

后续学习建议：

1. 结合 `simulink`搭建雷达系统级模型
2. 使用MATLAB Coder将核心算法转为C代码
3. 研究STAP（空时自适应处理）等先进雷达技术
4. 探索深度学习方法在SAR目标识别中的应用

（如需特定方向的深入扩展，请提供具体需求）