1.已知3阶椭圆IIR数字低通滤波器的性能指标为：通带截止频率0.4π，通带波纹为0.6dB，最小阻带衰减为32dB。设计一个6阶全通滤波器对其通带的群延时进行均衡。绘制低通滤波器和级联滤波器的群延时。

```
import Pkg
# 自动检测并安装缺失的包 (DSP, Optim)
# TyPlot 通常在 MWorks 中预装，但为了保险起见也可以检查，不过这里主要解决 Optim
required_packages = ["DSP", "Optim"]
for pkg in required_packages
    if Base.find_package(pkg) === nothing
        println("正在安装 $pkg ...")
        Pkg.add(pkg)
    end
end

using DSP
using TyPlot # 替换 Plots 为 TyPlot
using Optim
using LinearAlgebra
using Statistics # 确保 mean 和 std 可用

# ==========================================
# 1. 设计 3 阶椭圆低通滤波器 (IIR)
# ==========================================
println("正在设计椭圆滤波器...")

# 性能指标
wp = 0.4          # 通带截止频率 (归一化频率, 1.0对应π)
ws = 0.5          # 阻带起始频率
Rp = 0.6          # 通带波纹 (dB)
Rs = 32.0         # 最小阻带衰减 (dB)
N_iir = 3         # 滤波器阶数

# 设计滤波器对象
# 显式使用 DSP.Elliptic 以避免与 TyMath.Elliptic 冲突
response_type = Lowpass(wp)
design_method = DSP.Elliptic(N_iir, Rp, Rs)
iir_filter = digitalfilter(response_type, design_method)

# ==========================================
# 2. 设计 6 阶全通均衡器 (优化过程)
# ==========================================
println("正在优化全通均衡器参数 (可能需要几秒钟)...")

# 均衡器阶数 (6阶 = 3个二阶节)
N_ap = 6
n_sections = div(N_ap, 2)

# 定义频率网格 (只关注通带内的延时平坦度)
# 使用 0 到 0.4π 之间的频率点进行优化
w_pass = range(0, stop=wp*pi, length=100)

# 计算原始 IIR 滤波器的群延时
tau_iir = grpdelay(iir_filter, w_pass)

# 辅助函数：根据极点半径(r)和角度(theta)构建全通滤波器对象
function make_allpass(params)
    total_filter = nothing
    for i in 1:n_sections
        r = params[2*i - 1]
        theta = params[2*i]

        p = r * exp(im * theta)
        poles = [p, conj(p)]
        zeros = [1/conj(p), 1/p]

        section = ZeroPoleGain(zeros, poles, 1.0)

        if total_filter === nothing
            total_filter = section
        else
            total_filter = total_filter * section
        end
    end
    return total_filter
end
```

```
# 目标函数：计算总群延时的标准差 (越小越平坦)
function cost_function(params)
    try
        ap_filter = make_allpass(params)
        tau_ap = grpdelay(ap_filter, w_pass)
        tau_total = tau_iir + tau_ap
        return std(tau_total)
    catch
        return Inf
    end
end

# 初始猜测参数 [r, theta, r, theta, ...]
initial_params = [0.5, 0.1*pi, 0.6, 0.2*pi, 0.7, 0.3*pi]

# 设置参数边界
lower_bounds = repeat([0.0, 0.0], n_sections)
upper_bounds = repeat([0.99, pi], n_sections)

# 执行优化
res = optimize(cost_function, lower_bounds, upper_bounds, initial_params, Fminbox(BFGS()), Optim.Options(time_limit=10.0))
best_params = Optim.minimizer(res)

println("优化完成。")
println("最佳全通极点参数 (r, theta): ", round.(best_params, digits=3))

# ==========================================
# 3. 结果计算与绘图 (TyPlot 版本)
# ==========================================

# 构建最终的全通滤波器
ap_filter_final = make_allpass(best_params)

# 在更宽的频率范围内计算响应以便绘图
w_plot = range(0, stop=wp*pi, length=500)
tau_iir_plot = grpdelay(iir_filter, w_plot)
tau_ap_plot = grpdelay(ap_filter_final, w_plot)
tau_total_plot = tau_iir_plot + tau_ap_plot

# 准备绘图数据
x_data = w_plot ./ pi # 归一化频率

# 计算Y轴范围以便绘制垂直线
y_min = minimum([minimum(tau_iir_plot), minimum(tau_total_plot)])
y_max = maximum([maximum(tau_iir_plot), maximum(tau_total_plot)])

# 使用 TyPlot 绘图
# TyPlot 通常支持类似 plot(x, y, spec) 的语法
println("正在绘制结果...")

# 绘制原始低通滤波器群延时
# "b" 代表蓝色
plot(x_data, tau_iir_plot, "b", label="Original Lowpass")
hold("on") # 保持图像以叠加

# 绘制均衡后的群延时
# "r" 代表红色
plot(x_data, tau_total_plot, "r", label="Equalized (Total)")

# 绘制通带截止频率的垂直虚线
# 手动构造线段数据: ([wp, wp], [y_min, y_max])
# "k--" 代表黑色虚线
plot([wp, wp], [y_min, y_max], "k--", label="Cutoff Frequency")

title("Group Delay Equalization Result")
xlabel("Normalized Frequency (x pi rad/sample)")
ylabel("Group Delay (samples)")
legend() # 显示图例
grid("on") # 显示网格

println("平均群延时 (原始): ", round(mean(tau_iir_plot), digits=2))
println("平均群延时 (均衡后): ", round(mean(tau_total_plot), digits=2))
println("群延时波动 (原始 std): ", round(std(tau_iir_plot), digits=4))
println("群延时波动 (均衡后 std): ", round(std(tau_total_plot), digits=4))
```

2．设计巴特沃兹模拟低通滤波器，其滤波器的阶数和3-dB截止频率由键盘输入，程序能根据输入的参数，绘制滤波器的增益响应。

```
import Pkg
# 检查并添加必要的包 (如果尚未安装)
required_packages = ["TyPlot"]
for pkg in required_packages
    if Base.find_package(pkg) === nothing
        println("正在安装 $pkg ...")
        Pkg.add(pkg)
    end
end

using TyPlot
using LinearAlgebra

println("=== 巴特沃兹模拟低通滤波器设计 ===")

# ==========================================
# 1. 键盘输入参数 (优化版)
# ==========================================
function get_valid_input(prompt::String, parse_func::Function)
    while true
        print(prompt)
        flush(stdout) # 确保提示文字立即显示

        try
            # 读取一行并去除首尾空格
            input_str = strip(readline())

            # 如果是空行（比如误触回车），跳过本次循环继续等待
            if isempty(input_str)
                continue
            end

            # 尝试解析
            value = parse_func(input_str)
            return value
        catch
            # 解析失败时提示，而不是直接使用默认值退出
            println("输入格式不正确，请重新输入。")
        end
    end
end

try
    # 获取 N
    global N = get_valid_input("请输入滤波器阶数 N (整数, 例如 4): ", x -> parse(Int, x))

    # 获取 fc
    global fc = get_valid_input("请输入 3dB 截止频率 fc (Hz, 例如 1000): ", x -> parse(Float64, x))

    println("\n正在设计: 阶数 N = $N, 截止频率 fc = $fc Hz")

catch e
    # 只有在发生严重系统错误（如中断）时才捕获
    println("\n发生未知错误，程序终止。")
    rethrow(e)
end

# ==========================================
# 2. 滤波器设计 (计算极点)
# ==========================================
# 巴特沃兹滤波器的极点分布在 S 平面的圆上
# 归一化角频率 omega_c = 2 * pi * fc
wc = 2 * pi * fc

# 极点公式: sk = wc * exp(j * pi * (2k + N - 1) / 2N)
# k = 1 到 N
poles = ComplexF64[]
for k in 1:N
    theta = pi * (2 * k + N - 1) / (2 * N)
    s_k = wc * exp(im * theta)
    push!(poles, s_k)
end

println("\n计算得到的极点 (S平面):")
for (i, p) in enumerate(poles)
    println("p$i: $(round(real(p), digits=2)) + $(round(imag(p), digits=2))j")
```

```
end

# ==========================================
# 3. 计算频率响应
# ==========================================
# 定义频率范围用于绘图：从 0 到 3倍截止频率
f_plot = range(0, stop=3*fc, length=1000)
w_plot = 2 * pi .* f_plot

# 巴特沃兹模拟低通滤波器的传输函数 H(s) = wc^N / product(s - pk)
# 为了计算增益 |H(jw)|，我们将 s 替换为 jw

# 分子 (对于低通滤波器，DC增益为1，分子等于极点的乘积的模，即 wc^N)
numerator = wc^N

gains = Float64[]

for w in w_plot
    s = im * w
    # 分母 = (s - p1)*(s - p2)*...
    denominator = prod(s .- poles)

    # H(s) = Num / Den
    H = numerator / denominator

    # 存入幅值
    push!(gains, abs(H))
end

# 转换为 dB
gains_db = 20 .* log10.(gains)

# ==========================================
# 4. 绘制增益响应
# ==========================================
println("\n正在绘制增益响应...")

# 绘制幅频响应
plot(f_plot, gains_db, "b-", linewidth=2, label="Gain Response")
hold("on")

# 标记 -3dB 截止频率点
plot([fc, fc], [-60, 5], "r--", label="Cutoff Frequency (-3dB)")
plot(fc, -3.01, "ro") # 在曲线上标记点 (理论值约为 -3.01 dB)

title("Butterworth Analog Lowpass Filter Response (N=$N, fc=$(Int(fc))Hz)")
xlabel("Frequency (Hz)")
ylabel("Magnitude (dB)")
grid("on")
legend()

# 设置Y轴范围使图表更清晰
ylim(-40, 5)

println("绘图完成。")
```

3.已知系统的系统函数为：$H(z)=\frac{1-0.2z^{-1}+0.5z^{-2}}{1+3.2z^{-1}+1.5z^{-2}-0.8z^{-3}+1.4z^{-4}}$，用 MATLAB 进行部分分式展开，并写出展开后的表达式。

```
import Pkg
# 检查并安装 Polynomials 包，用于多项式运算
if Base.find_package("Polynomials") === nothing
    println("正在安装 Polynomials 包...")
    Pkg.add("Polynomials")
end

using Polynomials
using LinearAlgebra
using Printf

println("=== 系统函数部分分式展开 (ResidueZ) ===")

# ==========================================
# 1. 定义系统参数
# ==========================================
# H(z) = B(z) / A(z)
# 分子系数 (对应 1, z^-1, z^-2 ...)
b = [1.0, -0.2, 0.5]
```

```
# 分母系数 (对应 1, z^-1, z^-2, z^-3, z^-4)
a = [1.0, 3.2, 1.5, -0.8, 1.4]

println("分子系数 b: ", b)
println("分母系数 a: ", a)

# ==========================================
# 2. 计算极点 (Poles)
# ==========================================
# 将 H(z) 上下同乘 z^4 (分母最高阶)，转换为 z 的正幂次多项式
# A(z) = 1 + 3.2z^-1 + ... + 1.4z^-4
# -> A_poly(z) = z^4 + 3.2z^3 + 1.5z^2 - 0.8z + 1.4
# 注意：Polynomials 包默认系数顺序为升幂 [a0, a1, a2...]
# 我们的 a 向量是降幂排列 (对应 z^0, z^-1...)，且转换后对应 z^N, z^(N-1)...
# 因此 A_poly 的系数应该是 reverse(a)

den_coeffs = reverse(a) # [1.4, -0.8, 1.5, 3.2, 1.0]
den_poly = Polynomial(den_coeffs)

# 极点 p 是分母的根
# 显式使用 Polynomials.roots 避免与 TyMath.roots 冲突
p = Polynomials.roots(den_poly)

println("\n计算得到的极点 p:")
for (i, val) in enumerate(p)
    @printf("p%d = %.4f %+.4fim\n", i, real(val), imag(val))
end

# ==========================================
# 3. 计算留数 (Residues)
# ==========================================
# 对于 H(z) = B(z)/A(z)，我们需要找到 r_i 使得 H(z) = sum( r_i / (1 - p_i*z^-1) )
# 这等价于对 H(z)/z 进行部分分式展开：H(z)/z = sum( r_i / (z - p_i) )
# r_i = [H(z)/z * (z - p_i)] | z=p_i
#     = Num(p_i) / Den'(p_i)  (对于 H(z)/z 的分子分母)

# 构造 H(z)/z 的分子和分母多项式
# 原 H(z) = (z^4 - 0.2z^3 + 0.5z^2) / (z^4 + ...)  [分子补齐 z^2 以匹配分母阶数]
# H(z)/z = (z^3 - 0.2z^2 + 0.5z) / (z^4 + 3.2z^3 + ...)

# 构造分子系数 (升幂): 0.5z, -0.2z^2, 1.0z^3. 常数项为0
# b 向量是 [1.0, -0.2, 0.5]，对应 1 - 0.2z^-1 + 0.5z^-2
# 乘 z^4 变成 z^4 - 0.2z^3 + 0.5z^2
# 除 z 变成 z^3 - 0.2z^2 + 0.5z
# 系数(升幂): [0.0, 0.5, -0.2, 1.0]
num_coeffs_for_residue = [0.0, 0.5, -0.2, 1.0]
num_poly_res = Polynomial(num_coeffs_for_residue)

# 分母多项式即为之前的 den_poly (A_poly)
# r_i = Num_res(p_i) / Den_poly'(p_i)

# 显式使用 Polynomials.derivative
den_derivative = Polynomials.derivative(den_poly)

r = ComplexF64[]
for pole in p
    # 计算留数
    res_val = num_poly_res(pole) / den_derivative(pole)
    push!(r, res_val)
end

println("\n计算得到的留数 r:")
for (i, val) in enumerate(r)
    @printf("r%d = %.4f %+.4fim\n", i, real(val), imag(val))
end

# ==========================================
# 4. 生成展开表达式
# ==========================================
println("\n=== 展开后的表达式 ===")
println("H(z) = ")
for i in 1:length(r)
    r_re = real(r[i])
    r_im = imag(r[i])
    p_re = real(p[i])
    p_im = imag(p[i])

    # 格式化 r
    r_str = ""
    if abs(r_im) < 1e-6
        r_str = @sprintf("%.4f", r_re)
    else
```

```
        r_str = @sprintf("(%.4f %+.4fj)", r_re, r_im)
    end

    # 格式化 p
    p_str = ""
    if abs(p_im) < 1e-6
        p_str = @sprintf("1 - %.4fz^-1", p_re)
    else
        p_str = @sprintf("1 - (%.4f %+.4fj)z^-1", p_re, p_im)
    end

    # 打印项
    if i == 1
        print("      ", r_str, " / [", p_str, "]")
    else
        print("\n    + ", r_str, " / [", p_str, "]")
    end
end
println("\n")
```

4.设计切比雪夫 I 型 IIR 数字高通滤波器，其性能指标为：通带波纹 $\alpha_p = 0.5dB$，最小阻带衰减 $\alpha_s = 43dB$，通带和阻带边缘频率 $\omega_p = 0.75\pi$ rad 和 $\omega_s = 0.35\pi$ rad。绘制所设计的滤波器增益响应。。

```
import Pkg
# 检查并安装 DSP 包
required_packages = ["DSP", "TyPlot"]
for pkg in required_packages
    if Base.find_package(pkg) === nothing
        println("正在安装 $pkg ...")
        Pkg.add(pkg)
    end
end

using DSP
using TyPlot
using Printf

println("=== 切比雪夫 I 型 IIR 高通滤波器设计 ===")

# ========================================
# 1. 定义性能指标
# ========================================
Rp = 0.5          # 通带波纹 (dB)
Rs = 43.0         # 最小阻带衰减 (dB)
wp = 0.75         # 通带边缘频率 (归一化, 1.0 = π)
ws = 0.35         # 阻带边缘频率 (归一化, 1.0 = π)

println("指标参数:")
println("  通带波纹 Rp: $Rp dB")
println("  阻带衰减 Rs: $Rs dB")
println("  通带频率 wp: $(wp)π")
println("  阻带频率 ws: $(ws)π")

# ========================================
# 2. 计算最小阶数 N
# ========================================
# 为了确定满足衰减指标的最小阶数，我们需要使用模拟原型的公式
# 步骤:
# 1. 将数字频率预畸变转换为模拟频率 Omega = tan(w/2)
# 2. 计算高通滤波器的选择性因子 k = Omega_stop_analog / Omega_pass_analog (注意高通反转)
#    对于高通设计，我们将其映射到低通原型，变换比为 Omega_p / Omega
#    因此选择性因子 k_lp = (tan(wp*pi/2) / tan(ws*pi/2)) 的倒数?
#    标准低通原型的阻带边缘 Omega_s' = tan(wp*pi/2) / tan(ws*pi/2)

# 数字角频率
w_p_rad = wp * pi
w_s_rad = ws * pi

# 预畸变 (Pre-warping)
Omega_p = tan(w_p_rad / 2)
Omega_s = tan(w_s_rad / 2)

# 计算低通原型的归一化阻带频率 lambda_s
# 对于高通滤波器，通带映射到 1，阻带映射到 Omega_p / Omega_s
```

```
lambda_s = Omega_p / Omega_s

println("\n中间计算结果:")
@printf("  模拟通带频率 Ωp: %.4f\n", Omega_p)
@printf("  模拟阻带频率 Ωs: %.4f\n", Omega_s)
@printf("  低通原型阻带比 λs: %.4f\n", lambda_s)

# 切比雪夫阶数公式
# N >= acosh( sqrt((10^(0.1*Rs) - 1) / (10^(0.1*Rp) - 1)) ) / acosh(lambda_s)
numerator = acosh(sqrt((10^(0.1 * Rs) - 1) / (10^(0.1 * Rp) - 1)))
denominator = acosh(lambda_s)
N_exact = numerator / denominator
N = ceil(Int, N_exact)

println("  计算出的最小阶数 N: $N (精确值: $(round(N_exact, digits=3)))")

# ==========================================
# 3. 设计滤波器
# ==========================================
# 使用 DSPJl 进行设计
# 注意：DSPJl 的 Highpass(Wn) 中的 Wn 是通带截止频率
# Chebyshev1(N, ripple) 中的 ripple 是通带波纹

response_type = Highpass(wp)
design_method = Chebyshev1(N, Rp)

try
    global filter_obj = digitalfilter(response_type, design_method)
    println("\n滤波器设计成功！")
catch e
    println("\n滤波器设计失败: ", e)
end

# ==========================================
# 4. 计算频率响应并绘图
# ==========================================
# 定义频率向量 (0 到 π)
w_range = range(0, stop=pi, length=512)
h_resp = freqz(filter_obj, w_range)

# 计算幅度 (dB)
mag_db = 20 * log10.(abs.(h_resp))
frequencies_normalized = w_range / pi

println("正在绘制增益响应...")

# 绘制幅频响应
plot(frequencies_normalized, mag_db, "b-", linewidth=2, label="Gain Response")
hold("on")

# 绘制指标限制框
# 1. 通带区域 (0.75 到 1.0): 增益应在 -0.5dB 到 0dB 之间
#    这里只画下限 -0.5dB
plot([wp, 1.0], [-Rp, -Rp], "g--", linewidth=2, label="Passband Ripple Limit (-0.5dB)")

# 2. 阻带区域 (0 到 0.35): 增益应小于 -43dB
plot([0, ws], [-Rs, -Rs], "r--", linewidth=2, label="Stopband Attenuation Limit (-43dB)")

# 绘制截止频率标记
plot([wp, wp], [-60, 5], "k:", label="Passband Edge (0.75\\pi)")
plot([ws, ws], [-60, 5], "k:", label="Stopband Edge (0.35\\pi)")

# 设置图形属性
title("Chebyshev Type I Highpass Filter Response (N=$N)")
xlabel("Normalized Frequency (x \\pi rad/sample)")
ylabel("Magnitude (dB)")
xlim(0, 1)
ylim(-80, 5) # 设置合适的 Y 轴范围以显示衰减
grid("on")
legend()

println("绘图完成。")
```

5.已知复指数序列为: $x[n] = 0.2e^{(0.4+j0.5)n}$, 绘制 30 点该序列的实部和虚部。

```
import Pkg
# 检查并安装 TyPlot (如果尚未安装)
if Base.find_package("TyPlot") === nothing
    println("正在安装 TyPlot ...")
    Pkg.add("TyPlot")
end

using TyPlot

println("=== 复指数序列绘制 ===")

# ==========================================
# 1. 定义序列参数
# ==========================================
# 序列范围 n = 0 到 29 (30点)
n = 0:29

# 定义复指数序列 x[n] = 0.2 * e^((0.4 + j0.5)n)
# 在 Julia 中，虚数单位用 im 表示
# 注意使用 .进行广播运算
alpha = 0.4 + 0.5im
x = 0.2 * exp.(alpha .* n)

# ==========================================
# 2. 提取实部和虚部
# ==========================================
x_real = real(x)
x_imag = imag(x)

println("计算完成。准备绘图...")

# ==========================================
# 3. 绘制图形
# ==========================================
# 使用 subplot 将实部和虚部画在同一张图的两个子图中

# --- 子图 1: 实部 ---
subplot(2, 1, 1)
# stem 用于绘制离散序列 (火柴杆图)
# 如果 TyPlot 的 stem 参数与 MATLAB 类似，可以直接传入 x, y
stem(n, x_real, "b")
title("Real Part of x[n]")
ylabel("Amplitude")
grid("on")

# --- 子图 2: 虚部 ---
subplot(2, 1, 2)
stem(n, x_imag, "r")
title("Imaginary Part of x[n]")
xlabel("n (Time Index)")
ylabel("Amplitude")
grid("on")

println("绘图完成。")
```

6.设计切比雪夫 I 型模拟低通滤波器, 其滤波器的阶数, 3-dB 截止频率和通带的波纹由键盘输入, 程序能根据输入的参数, 绘制滤波器的增益响应。

```
import Pkg
# 检查并添加 TyPlot
if Base.find_package("TyPlot") === nothing
    println("正在安装 TyPlot ...")
    Pkg.add("TyPlot")
end

using TyPlot
using LinearAlgebra

println("=== 切比雪夫 I 型模拟低通滤波器设计 ===")
```

```
# ========================================
# 1. 键盘输入参数
# ========================================
function get_valid_input(prompt::String, parse_func::Function)
    while true
        print(prompt)
        flush(stdout)
        try
            input_str = strip(readline())
            if isempty(input_str) continue end
            return parse_func(input_str)
        catch
            println("输入格式不正确，请重新输入。")
        end
    end
end

try
    # 获取参数
    global N = get_valid_input("请输入滤波器阶数 N (整数, 例如 4): ", x -> parse(Int, x))
    global fc = get_valid_input("请输入截止频率 fc (Hz, 例如 1000): ", x -> parse(Float64, x))
    global Rp = get_valid_input("请输入通带波纹 Rp (dB, 例如 0.5): ", x -> parse(Float64, x))

    println("\n正在设计: N = $N, fc = $fc Hz, Rp = $Rp dB")

catch e
    println("\n错误: $e")
    exit()
end

# ========================================
# 2. 计算极点 (S平面)
# ========================================
# 切比雪夫 I 型滤波器的极点位于 S 平面的椭圆上
# 这里的 fc 被视为通带边缘频率 (Passband Edge Frequency)

wc = 2 * pi * fc
epsilon = sqrt(10^(Rp/10) - 1) # 波纹因子
mu = asinh(1/epsilon) / N      # 辅助参数

poles = ComplexF64[]
for k in 1:N
    # 角度定义
    # theta_k = pi/2 + (2k-1)/(2N) * pi
    # 注意: 有些定义 k 从 0 到 N-1，这里 k 从 1 到 N，公式对应调整
    theta = pi/2 + ((2*k - 1) * pi) / (2 * N)

    # 极点公式: s_k = wc * (-sinh(mu)sin(theta) + j cosh(mu)cos(theta))
    real_part = -sinh(mu) * sin(theta)
    imag_part = cosh(mu) * cos(theta)
    s_k = wc * (real_part + im * imag_part)

    push!(poles, s_k)
end

println("\n计算得到的极点 (S平面):")
for (i, p) in enumerate(poles)
    println("p$i: $(round(real(p), digits=2)) + $(round(imag(p), digits=2))j")
end

# ========================================
# 3. 计算频率响应
# ========================================
f_plot = range(0, stop=3*fc, length=1000)
w_plot = 2 * pi .* f_plot

# 确定分子系数 K
# 切比雪夫 I 型滤波器在 w=0 处的幅值为:
# - 如果 N 是奇数: |H(0)| = 1 (0 dB)
# - 如果 N 是偶数: |H(0)| = 1 / sqrt(1 + epsilon^2) (-Rp dB)

# 计算分母多项式在 s=0 处的值 (即所有极点的积的模)
denom_at_0 = abs(prod(-poles)) # s=0 -> product(-pk)

target_dc_gain = (N % 2 == 1) ? 1.0 : (1.0 / sqrt(1 + epsilon^2))
numerator = target_dc_gain * denom_at_0

gains = Float64[]

for w in w_plot
    s = im * w
    # H(s) = K / product(s - pk)
```

```
    denominator = prod(s .- poles)
    H = numerator / denominator
    push!(gains, abs(H))
end

gains_db = 20 .* log10.(gains)

# ========================================
# 4. 绘图
# ========================================
println("\n正在绘制增益响应...")

plot(f_plot, gains_db, "b-", linewidth=2, label="Gain Response")
hold("on")

# 绘制通带波纹下限
plot([0, fc], [-Rp, -Rp], "g--", label="Passband Ripple (-$Rp dB)")

# 标记截止频率点
plot([fc, fc], [-60, 5], "r:", label="Cutoff Frequency")

title("Chebyshev Type I Analog Filter (N=$N, fc=$(Int(fc))Hz, Rp=$(Rp)dB)")
xlabel("Frequency (Hz)")
ylabel("Magnitude (dB)")
grid("on")
legend()

# 调整 Y 轴范围以便观察波纹
ylim(min(-40, -Rp-10), max(5, Rp+2))

println("绘图完成。")
```

7. 已知系统的系统函数为：$H(z)=0.2+\frac{1}{1+3.2z^{-1}}+\frac{0.6}{1-2.4z^{-1}}+\frac{1.8}{(1-2.4z^{-1})^2}$。用 MATLAB 求系统 z 变换的有理形式，并写出有理形式的表达式。

```
# 引入包管理器
import Pkg

# 尝试引入 SymPy，如果报错则自动安装
try
    using SymPy
catch e
    println("正在安装 SymPy 包，请稍候...")
    Pkg.add("SymPy")
    using SymPy
end

# 定义符号 z
@vars z

# 定义 z^-1 (为了方便书写公式)
inv_z = z^(-1)

# 输入题目给定的系统函数 H(z)
# H(z) = 0.2 + 1/(1+3.2z^-1) + 0.6/(1-2.4z^-1) + 1.8/((1-2.4z^-1)^2)
H = 0.2 + 1/(1 + 3.2*inv_z) + 0.6/(1 - 2.4*inv_z) + 1.8/((1 - 2.4*inv_z)^2)

# 计算有理形式 (通分合并)
# together 函数会将多项式合并到同一个分母上
H_rational = together(H)

# 进一步简化表达式（展开分子分母）
# 这一步通常会将结果整理为 z 的正幂次形式
H_simplified = simplify(H_rational)

# 获取分子和分母
num = numer(H_simplified)
den = denom(H_simplified)

# 打印结果
println("----------- 计算结果 -----------")
println("原始表达式 H(z):")
println(H)
println("\n合并后的有理形式 H_rational(z):")
println(H_simplified)
```

```
println("\n---------- 分子与分母 ----------")
println("分子 N(z): ", num)
println("分母 D(z): ", den)

# 如果需要转换回 z^-1 的形式 (DSP 中常用)，可以上下同除以 z 的最高次幂
# 这里主要展示数学上的有理多项式形式
```

8.设计巴特沃兹 IIR 数字带通滤波器，其性能指标为：归一化通带截止频率为 $\omega_{p1} = 0.4\pi, \omega_{p2} = 0.6\pi$，归一化阻带截止频率为 $\omega_{s1} = 0.3\pi$，$\omega_{s2} = 0.7\pi$，通带波纹为 0.6dB，最小阻带衰减为 35dB。绘制所设计的滤波器增益响应。

```
import Pkg

# ------------------------------------------------------------
# 依赖检查
# ------------------------------------------------------------
# 检查并安装 DSP 库
try
    using DSP
catch
    println("正在安装 DSP 包...")
    Pkg.add("DSP")
    using DSP
end

# 加载 TyPlot (MWorks 原生绘图库)
using TyPlot

# ------------------------------------------------------------
# 1. 定义滤波器性能指标
# ------------------------------------------------------------
# 归一化频率 (ω/π)
# 注意：DSPjl 的 buttord 函数要求带通频率范围必须是 Tuple (元组)，不能是 Vector [向量]
wp = (0.4, 0.6)     # 通带截止频率 (使用圆括号)
ws = (0.3, 0.7)     # 阻带截止频率 (使用圆括号)
Rp = 0.6            # 通带最大波纹 (dB)
Rs = 35             # 阻带最小衰减 (dB)

println("正在计算滤波器参数...")

# ------------------------------------------------------------
# 2. 计算巴特沃兹滤波器参数
# ------------------------------------------------------------
# buttord(wp::Tuple, ws::Tuple, Rp, Rs)
N, Wn = buttord(wp, ws, Rp, Rs)

println("---------- 设计结果 ----------")
println("滤波器类型: Butterworth IIR Bandpass")
println("计算得到的阶数 N: ", N)
println("3dB 截止频率 Wn: ", Wn)

# ------------------------------------------------------------
# 3. 设计滤波器
# ------------------------------------------------------------
# Wn 返回的也是 Tuple，直接通过索引访问
responsetype = Bandpass(Wn[1], Wn[2])
designmethod = Butterworth(N)
filter_system = digitalfilter(responsetype, designmethod)

# ------------------------------------------------------------
# 4. 分析频率响应
# ------------------------------------------------------------
num_points = 1024
w_vals = range(0, π, length=num_points)

# 计算频率响应
H = freqz(filter_system, w_vals)
mag_db = 20 * log10.(abs.(H))

# x轴数据 (归一化频率)
x_axis = w_vals / π

# ------------------------------------------------------------
# 5. 使用 TyPlot 绘制增益响应曲线
# ------------------------------------------------------------
```

```
# 创建图形窗口
figure("Butterworth IIR Bandpass Filter")

# 绘制主响应曲线
plot(x_axis, mag_db, "b-", label="Filter Response", linewidth=2)

# 添加网格 (修改为 "on")
grid("on")

# 设置标题和轴标签
title("Butterworth IIR Bandpass Filter Gain Response")
xlabel("Normalized Frequency (×π rad/sample)")
ylabel("Magnitude (dB)")

# 保持当前图形以便添加辅助线 (修改为 "on")
hold("on")

# 添加指标辅助线 (通带 -0.6dB)
plot([0.4, 0.6], [-Rp, -Rp], "g--", label="Passband Spec (-0.6dB)", linewidth=1.5)

# 添加指标辅助线 (阻带 -35dB)
# 分两段画，避免中间连线
plot([0.0, 0.3], [-Rs, -Rs], "r--", label="Stopband Spec (-35dB)", linewidth=1.5)
plot([0.7, 1.0], [-Rs, -Rs], "r--", linewidth=1.5)

# 设置Y轴范围
ylim(-80, 5)

# 显示图例
legend()

println("绘图完成。")
```

9.已知指数序列为: $x[n] = 2(0.9)^n$, 绘制 24 点该序列。

```
using TyPlot

# 1. 定义序列范围
# n 从 0 到 23，共 24 个点
n = 0:23

# 2. 计算指数序列 x[n] = 2 * (0.9)^n
# 注意：在 Julia 中，对向量进行幂运算需要使用点运算符 .^
# 这里的 2 .* ... 表示标量与向量的每个元素相乘
x = 2 .* (0.9) .^ n

# 3. 绘制图形
figure("Discrete Exponential Sequence")

# 使用 stem 绘制离散序列 (火柴杆图)
# 语法与 MATLAB 类似
stem(n, x)

# 添加标题和坐标轴标签
# TyPlot 支持 LaTeX 格式的数学公式渲染
title("Exponential Sequence: \$x[n] = 2(0.9)^{n}\$")
xlabel("n (Sample Index)")
ylabel("Amplitude x[n]")

# 开启网格
grid("on")

# 设置 X 轴范围稍微宽一点，以便看清首尾的点
xlim(-1, 24)

println("绘图完成。")
```

10.设计椭圆模拟低通滤波器, 其滤波器的阶数, 3-dB 截止频率, 通带的波纹和阻带衰减由键盘输入, 程序能根据输入的参数, 绘制滤波器的增益响应。

```
# 导入必要的库
using DSP
using TyPlot
```

```
"""
MWorks 椭圆低通滤波器设计工具
功能:
1. 通过键盘接收用户输入的滤波器参数
2. 计算椭圆滤波器的频率响应
3. 使用 TyPlot 绘制增益响应 (Bode Plot Magnitude)
"""
function design_and_plot_elliptic_filter()
    # ==============================
    # 1. 获取用户输入
    # ==============================
    println("=== 椭圆低通滤波器设计 (MWorks) ===")

    try
        # 强制刷新输出缓冲区，确保提示词在输入框之前显示
        flush(stdout)

        print("请输入滤波器阶数 (整数, 例如 4): ")
        input_str = readline()
        if isempty(input_str); println("输入为空"); return; end
        N = parse(Int, input_str)

        print("请输入采样频率 Fs (Hz, 例如 1000): ")
        input_str = readline()
        if isempty(input_str); println("输入为空"); return; end
        Fs = parse(Float64, input_str)

        print("请输入截止频率 Fc (Hz, 必须小于 Fs/2, 例如 200): ")
        input_str = readline()
        if isempty(input_str); println("输入为空"); return; end
        Fc = parse(Float64, input_str)

        print("请输入通带波纹 Rp (dB, 例如 1.0): ")
        input_str = readline()
        if isempty(input_str); println("输入为空"); return; end
        Rp = parse(Float64, input_str)

        print("请输入阻带衰减 Rs (dB, 例如 40.0): ")
        input_str = readline()
        if isempty(input_str); println("输入为空"); return; end
        Rs = parse(Float64, input_str)

        # 验证 Nyquist 频率限制
        if Fc >= Fs / 2
            println("错误：截止频率必须小于采样频率的一半 (Nyquist 频率)。")
            return
        end

        # ==============================
        # 2. 滤波器设计
        # ==============================
        println("\n正在计算滤波器系数...")

        # [修复] 使用 DSP.Lowpass 明确指定包名
        response_type = DSP.Lowpass(Fc; fs=Fs)

        # [修复] 使用 DSP.Elliptic 明确指定包名，解决 UndefVarError 问题
        design_method = DSP.Elliptic(N, Rp, Rs)

        # [修复] 使用 DSP.digitalfilter
        filter_object = DSP.digitalfilter(response_type, design_method)

        # ==============================
        # 3. 计算频率响应
        # ==============================
        # 生成频率点：从 0 到 Nyquist 频率，取 1024 个点
        freqs_range = range(0, Fs/2, length=1024)

        # [修复] 使用 DSP.freqz
        h = DSP.freqz(filter_object, freqs_range, Fs)

        # 计算幅值增益 (dB)
        magnitude_db = 20 * log10.(abs.(h))

        # ==============================
        # 4. 使用 TyPlot 绘图
        # ==============================
        println("正在绘制增益响应曲线...")

        # 清除当前图形 (如果有)
        TyPlot.clf()
```

```
        # 绘制曲线
        TyPlot.plot(freqs_range, magnitude_db, "b-", linewidth=1.5, label="Gain Response")

        # 设置标题和标签
        TyPlot.title("椭圆低通滤波器增益响应 (Elliptic Lowpass Filter)")
        TyPlot.xlabel("频率 (Hz)")
        TyPlot.ylabel("增益 (dB)")

        # 添加网格
        TyPlot.grid(true)

        # 添加图例
        TyPlot.legend()

        println("绘图完成！")

    catch e
        # 打印详细错误栈，方便调试
        println("运行出错: ", e)
        # showerror(stdout, e, catch_backtrace()) # 如果需要更详细的堆栈信息可以取消注释
        println("\n提示：请检查 DSP 包是否已安装 (import Pkg; Pkg.add(\"DSP\"))")
    end
end

# 运行主函数
design_and_plot_elliptic_filter()
```

11.已知系统的系统函数为: $H(z)=\frac{1-0.2z^{-1}+0.5z^{-2}}{1+3.2z^{-1}+1.5z^{-2}-0.8z^{-3}+1.4z^{-4}}$ 。用 MATLAB 的 `impz` 函数求 h[n] 的前 30 个样本值。

```
using DSP
using TyPlot

"""
计算并绘制离散系统的脉冲响应
系统函数: H(z) = (1 - 0.2z^-1 + 0.5z^-2) / (1 + 3.2z^-1 + 1.5z^-2 - 0.8z^-3 + 1.4z^-4)
"""
function solve_impulse_response()
    # 1. 定义系统系数
    # 分子系数 b (对应 z^-0, z^-1, z^-2)
    b = [1.0, -0.2, 0.5]

    # 分母系数 a (对应 z^-0, z^-1, z^-2, z^-3, z^-4)
    a = [1.0, 3.2, 1.5, -0.8, 1.4]

    # 2. 生成输入信号：单位脉冲 (Unit Impulse)
    # 我们需要前 30 个样本
    N = 30
    delta = zeros(Float64, N)
    delta[1] = 1.0  # 在 n=0 处为 1 (Julia 索引从 1 开始，所以对应数组第 1 个元素)

    # 3. 计算脉冲响应
    # 使用 DSP.filt 函数对单位脉冲进行滤波，得到的输出即为脉冲响应 h[n]
    h = DSP.filt(b, a, delta)

    # 4. 打印数值结果
    println("=== 脉冲响应 h[n] 前 30 个样本值 ===")
    for i in 1:N
        # n 从 0 开始，数组索引 i 从 1 开始
        println("n = $(i-1):  $(h[i])")
    end

    # 5. 绘制图形 (离散序列通常使用杆图 stem，这里用带标记的线图模拟)
    TyPlot.clf()

    # 创建时间轴 n = 0 到 29
    n_axis = 0:(N-1)

    # 绘制
    TyPlot.plot(n_axis, h, "bo-", linewidth=1, markersize=5, label="h[n]")

    # 设置图表属性
    TyPlot.title("系统脉冲响应 h[n] (Impulse Response)")
    TyPlot.xlabel("样本序号 n")
    TyPlot.ylabel("幅值")
    TyPlot.grid(true)
    TyPlot.legend()
```

```
    println("\n计算与绘图完成。")
end

# 运行函数
solve_impulse_response()
```

12.已知 5 阶椭圆 IIR 数字低通滤波器的性能指标为: 通带截止频率 0.35π, 通带波纹为 0.8dB, 最小阻带衰减为 35dB。设计一个 10 阶全通滤波器对其通带的群延时进行均衡。绘制低通滤波器和级联滤波器的群延时。

```
using DSP
using TyPlot

"""
椭圆滤波器设计与群延时分析
任务:
1. 设计 5 阶椭圆低通滤波器 (0.35pi, 0.8dB ripple, 35dB attenuation)
2. 框架性展示 10 阶全通滤波器级联 (注：自动系数优化需额外算法支持)
3. 绘制群延时对比图
"""
function analyze_group_delay()
    println("=== 滤波器群延时分析 ===")

    # ==============================
    # 1. 设计椭圆低通滤波器 (Lowpass)
    # ==============================
    # 规格: N=5, Fp=0.35pi (归一化频率 0.35), Rp=0.8dB, Rs=35dB
    N_lp = 5
    Wn = 0.35  # 归一化频率 (1.0 = Nyquist, 即 pi)
    Rp = 0.8
    Rs = 35.0

    println("1. 设计椭圆低通滤波器: Order=$N_lp, Wn=$Wn, Rp=$Rp dB, Rs=$Rs dB")

    # 设计滤波器
    # DSP.Lowpass(Wn) 中的 Wn 是归一化频率 (0~1)
    lp_responsetype = DSP.Lowpass(Wn)
    lp_designmethod = DSP.Elliptic(N_lp, Rp, Rs)
    lp_filter = DSP.digitalfilter(lp_responsetype, lp_designmethod)

    # ==============================
    # 2. 构建全通滤波器 (Allpass Equalizer)
    # ==============================
    # 目标：设计 10 阶全通滤波器均衡通带群延时

    N_ap = 10
    println("2. 初始化全通滤波器结构 (Order=$N_ap)")

    # --- [占位符系数] ---
    # 目前设置为 [1.0, 0, ...] 代表直通
    a_ap = zeros(Float64, N_ap + 1)
    a_ap[1] = 1.0

    # 全通滤波器的分子系数是分母系数的倒序
    b_ap = reverse(a_ap)

    # 创建全通滤波器对象
    ap_filter = DSP.PolynomialRatio(b_ap, a_ap)

    # ==============================
    # 3. 计算群延时 (Group Delay)
    # ==============================
    println("3. 计算群延时特性...")

    # 频率点数量
    n_points = 512

    # [修复] 显式生成频率向量 (0 到 pi)，避免 grpdelay 返回标量导致的 BoundsError
    # range 返回的是一个迭代器，collect 转换为数组
    w_rad = collect(range(0, π, length=n_points))

    # 计算低通滤波器的群延时
    # 当传入频率向量时，grpdelay 仅返回延时向量
    gd_lp = DSP.grpdelay(lp_filter, w_rad)
```

```
    # 计算全通滤波器的群延时
    gd_ap = DSP.grpdelay(ap_filter, w_rad)

    # 级联系统的群延时 = 低通延时 + 全通延时
    gd_total = gd_lp + gd_ap

    # [优化] 将频率归一化 (0~1) 以便与 Wn 比较和绘图
    w_norm = w_rad / π

    # 提取通带部分的索引用于绘图 (只看通带内部情况)
    # 因为 Wn 是 0.35 (归一化)，我们对比 w_norm
    passband_indices = w_norm .<= (Wn * 1.2) #稍微多画一点以便观察截止点

    # ==============================
    # 4. 绘制图形
    # ==============================
    println("4. 绘制群延时曲线...")
    TyPlot.clf() # 清除旧图

    # 绘制低通滤波器的群延时
    TyPlot.plot(w_norm[passband_indices], gd_lp[passband_indices], "b-", linewidth=2, label="Lowpass Filter Delay")

    # 绘制级联系统的群延时
    TyPlot.plot(w_norm[passband_indices], gd_total[passband_indices], "r--", linewidth=1.5, label="Cascaded (LP + AP) Delay")

    TyPlot.title("Group Delay in Passband")
    TyPlot.xlabel("Normalized Frequency (x pi rad/sample)")
    TyPlot.ylabel("Group Delay (samples)")
    TyPlot.grid(true)
    TyPlot.legend()

    # 标记截止频率
    # TyPlot.axvline(x=Wn, color="k", linestyle="--", label="Cutoff")

    println("完成！请查看绘图窗口。")
end

# 运行分析
analyze_group_delay()
```

13. 编写 4 点滑动平均滤波器程序。原始未受干扰的序列为：$s[n] = 3[n(0.8)^n]$，加性噪声信号 d[n] 为随机序列，幅度 0.6，受干扰的序列为：$x[n] = s[n] + d[n]$，分别绘制长度为 40 的原始未受干扰的序列，噪声序列和受干扰序列，以及滑动平均滤波器的输出。

```
using TyPlot
using Random
using DSP

"""
4 点滑动平均滤波器演示
1. 生成原始信号 s[n] = 3 * n * (0.8)^n
2. 生成噪声信号 d[n] (幅度 0.6)
3. 合成受干扰信号 x[n] = s[n] + d[n]
4. 使用 4 点滑动平均滤波器进行处理
5. 绘制所有信号波形
"""
function moving_average_demo()
    # ==============================
    # 1. 信号生成
    # ==============================
    N = 40              # 序列长度
    n = 0:(N-1)         # 时间索引 n = 0, 1, ..., 39

    # 原始信号 s[n]
    s = 3 .* n .* (0.8).^n

    # 噪声信号 d[n]
    # 设定随机种子以保证结果可重复
    Random.seed!(42)
    # 生成幅度为 0.6 的随机噪声，假设为均匀分布 [-0.6, 0.6]
    d = 1.2 .* rand(N) .- 0.6

    # 受干扰信号 x[n]
    x = s + d
```

```
    # ==============================
    # 2. 滤波器设计与应用
    # ==============================
    # 4 点滑动平均滤波器
    # 差分方程: y[n] = 1/4 * (x[n] + x[n-1] + x[n-2] + x[n-3])
    # 系统函数: H(z) = 0.25 + 0.25z^-1 + 0.25z^-2 + 0.25z^-3

    M = 4
    b = ones(M) / M   # 分子系数 [0.25, 0.25, 0.25, 0.25]
    a = [1.0]         # 分母系数 [1.0]

    # 使用 DSP.filt 进行滤波
    y = DSP.filt(b, a, x)

    # ==============================
    # 3. 绘图
    # ==============================
    println("正在绘制波形...")
    TyPlot.clf() # 清除旧图

    # 使用 2x2 子图布局

    # 子图 1: 原始信号 s[n]
    TyPlot.subplot(2, 2, 1)
    TyPlot.plot(n, s, "b.-", label="s[n]") # 蓝色点线
    TyPlot.title("原始信号 s[n]")
    TyPlot.grid(true)
    # TyPlot.ylabel("幅值")

    # 子图 2: 噪声信号 d[n]
    TyPlot.subplot(2, 2, 2)
    TyPlot.plot(n, d, "g.-", label="d[n]") # 绿色点线
    TyPlot.title("噪声 d[n]")
    TyPlot.grid(true)

    # 子图 3: 受干扰信号 x[n]
    TyPlot.subplot(2, 2, 3)
    TyPlot.plot(n, x, "r.-", label="x[n]") # 红色点线
    TyPlot.title("受干扰信号 x[n]")
    TyPlot.xlabel("样本 n")
    TyPlot.ylabel("幅值")
    TyPlot.grid(true)

    # 子图 4: 滤波输出 y[n]
    TyPlot.subplot(2, 2, 4)
    TyPlot.plot(n, y, "k.-", label="y[n]", linewidth=1.5) # 黑色加粗点线
    TyPlot.title("4点滑动平均输出 y[n]")
    TyPlot.xlabel("样本 n")
    TyPlot.grid(true)

    # 调整整体布局标题 (如果支持)
    # TyPlot.suptitle("滑动平均滤波器效果对比")

    println("处理完成。")
    println("原始信号 s[n] (前5个点): ", s[1:5])
    println("滤波输出 y[n] (前5个点): ", y[1:5])
end

# 运行主函数
moving_average_demo()
```

14.绘制长度为 10 点的矩形序列的 16 点离散傅立叶变换样本的幅度和相位。

```
using TyPlot
using FFTW

"""
DFT 分析演示
任务:
1. 生成长度为 10 的矩形序列
2. 计算 16 点 DFT (补零)
3. 绘制幅度和相位谱
"""
function dft_rectangular_window()
    println("=== 离散傅立叶变换 (DFT) 分析 ===")

    # ==============================
    # 1. 信号生成
```

```
    # ==============================
    # 长度为 10 的矩形序列 (Rectangular Sequence)
    # x[n] = 1, for 0 <= n < 10
    N_seq = 10
    x = ones(Float64, N_seq)

    println("原始序列长度: $N_seq")

    # ==============================
    # 2. 16 点 DFT 计算
    # ==============================
    N_fft = 16
    println("DFT 变换点数: $N_fft")

    # 构造补零后的序列
    # 将 x 放入长度为 16 的零数组的前 10 个位置
    x_padded = zeros(Float64, N_fft)
    x_padded[1:N_seq] = x

    # 计算 FFT
    # 注意：Julia 的 fft 函数位于 FFTW 包中
    X_k = fft(x_padded)

    # ==============================
    # 3. 计算幅度与相位
    # ==============================
    # 幅度 Spectrum Magnitude
    magnitude = abs.(X_k)

    # 相位 Spectrum Phase (弧度)
    phase = angle.(X_k)

    # ==============================
    # 4. 绘图
    # ==============================
    println("正在绘制幅度和相位谱...")
    TyPlot.clf() # 清除旧图

    # 频率轴 k = 0, 1, ..., 15
    k = 0:(N_fft-1)

    # --- 子图 1: 幅度谱 ---
    TyPlot.subplot(2, 1, 1)
    # 使用 "bo-" (蓝色圆点实线) 模拟离散频谱的杆状图效果
    TyPlot.plot(k, magnitude, "bo-", linewidth=1.5, markersize=5, label="|X[k]|")
    TyPlot.title("16点 DFT 幅度谱 (Magnitude)")
    TyPlot.xlabel("频率索引 k")
    TyPlot.ylabel("幅度")
    TyPlot.grid(true)
    # 调整 Y 轴范围以更美观地显示主瓣
    TyPlot.ylim(0, maximum(magnitude) * 1.1)

    # --- 子图 2: 相位谱 ---
    TyPlot.subplot(2, 1, 2)
    # 使用 "ro-" (红色圆点实线)
    TyPlot.plot(k, phase, "ro-", linewidth=1.5, markersize=5, label="∠X[k]")
    TyPlot.title("16点 DFT 相位谱 (Phase)")
    TyPlot.xlabel("频率索引 k")
    TyPlot.ylabel("相位 (rad)")
    TyPlot.grid(true)
    # 设置 Y 轴范围为 -pi 到 pi，方便观察
    TyPlot.ylim(-pi, pi)

    println("处理完成。")
end

# 运行分析函数
# 注意：如果提示 UndefVarError: `fft` not defined，请先运行 `import Pkg; Pkg.add("FFTW")`
try
    dft_rectangular_window()
catch e
    println("运行出错: ", e)
    println("请确保已安装 FFTW 包: import Pkg; Pkg.add(\"FFTW\")")
end
```

15.已知系统的系统函数为: $H(z)=\frac{1-0.2z^{-1}+0.5z^{-2}}{1+3.2z^{-1}+1.5z^{-2}-0.8z^{-3}+1.4z^{-4}}$ 。用 MATLAB 的 `filter` 函数求 h[n] 的前 20 个样本值。

```
using TyPlot
using DSP # Julia的信号处理包，包含filt函数

# 1. 定义系统参数
# H(z) = (1 - 0.2z^-1 + 0.5z^-2) / (1 + 3.2z^-1 + 1.5z^-2 - 0.8z^-3 + 1.4z^-4)
# 分子系数 b (对应 z^-0, z^-1, z^-2 ...)
b = [1.0, -0.2, 0.5]

# 分母系数 a (对应 z^-0, z^-1, z^-2, z^-3, z^-4)
a = [1.0, 3.2, 1.5, -0.8, 1.4]

# 2. 构造输入信号：单位脉冲序列 delta[n]
N = 20          # 样本数量
x = zeros(N)    # 初始化为全0
x[1] = 1.0      # 在 n=0 处 (索引1) 设置为 1

# 3. 计算冲激响应 h[n]
# 使用 filt 函数求解差分方程，等同于 MATLAB 的 filter(b, a, x)
h = filt(b, a, x)

# 打印数值结果
println("前 20 个样本值 h[n]:")
for i in 1:N
    # 打印格式：n=索引值, 样本值保留4位小数
    println("n=$(i-1): $(round(h[i], digits=4))")
end

# 4. 使用 TyPlot 绘图
# 使用 stem 函数绘制离散序列图 (如果 TyPlot 支持 stem)
# 如果环境不支持 stem，可以使用 plot(0:N-1, h, "o-")
figure("Impulse Response") # 创建图形窗口
stem(0:N-1, h, "b-o")      # 绘制火柴杆图 (蓝色，圆点)

title("系统冲激响应 h[n] (前20个点)")
xlabel("采样点 n")
ylabel("幅度")
grid(true)              # 显示网格
```

16.利用 Hermann 公式估计 FIR 低通滤波器的阶数。该滤波器的性能指标为: 通带截止频率为 1500Hz, 阻带截止频率为 1800Hz, 通带波纹为 $\delta_p = 0.015$, 阻带波纹为 $\delta_s = 0.021$, 抽样频率为 5000Hz。

```
using TyPlot

# 1. 定义滤波器指标
Fs = 5000.0     # 抽样频率 (Hz)
fp = 1500.0     # 通带截止频率 (Hz)
fs = 1800.0     # 阻带截止频率 (Hz)
dp = 0.015      # 通带波纹 (delta_p)
ds = 0.021      # 阻带波纹 (delta_s)

# 2. 预处理参数
# 计算归一化过渡带宽度 Delta F
# 注意：Herrmann 公式中的频率通常归一化为 Fs=1
Delta_F = (fs - fp) / Fs

# 计算对数值
log_dp = log10(dp)
log_ds = log10(ds)

# 3. 计算 Hermann 公式系数
# 系数定义 (参考 Herrmann, Schuessler, Dehner, 1973)
# D_inf = a(dp) * log10(ds) + g(dp)

# 计算 a(dp)
# a(dp) = 0.005309 * (log_dp)^2 + 0.07114 * log_dp - 0.4761
a_dp = 0.005309 * (log_dp^2) + 0.07114 * log_dp - 0.4761

# 计算 g(dp)
# g(dp) = -0.00266 * (log_dp)^2 - 0.5941 * log_dp - 0.4278
g_dp = -0.00266 * (log_dp^2) - 0.5941 * log_dp - 0.4278
```

```
# 计算 D_infinity
D_inf = a_dp * log_ds + g_dp

# 计算修正项 f(dp, ds)
# f(dp, ds) = 11.01217 + 0.51244 * (log_dp - log_ds)
f_correction = 11.01217 + 0.51244 * (log_dp - log_ds)

# 4. 计算滤波器长度 N
# N = (D_inf - f_correction * (Delta_F)^2) / Delta_F
N_exact = (D_inf - f_correction * (Delta_F^2)) / Delta_F

# 滤波器长度通常取向上取整
N = ceil(Int, N_exact)

# 滤波器阶数 M = N - 1
Order = N - 1

# 5. 输出结果
println("---------- Hermann 公式估算结果 ----------")
println("归一化过渡带宽度 ΔF: $(round(Delta_F, digits=4))")
println("D_infinity: $(round(D_inf, digits=4))")
println("计算得到的精确长度 N_exact: $(round(N_exact, digits=4))")
println("估算滤波器长度 N: $N")
println("估算滤波器阶数 M (N-1): $Order")
println("----------------------------------------")

# 6. (可选) 绘制规格示意图
# 这是一个简单的图示，用于直观展示通带和阻带的位置
figure("Filter Specifications")
# 绘制理想的幅频特性轮廓
# 修改变量名 freqs 为 freq_specs 以避免与 DSP.freqs 冲突
freq_specs = [0, fp, fs, Fs/2]
amps = [1, 1, 0, 0]
plot(freq_specs, amps, "r--")
title("FIR 低通滤波器设计规格")
xlabel("频率 (Hz)")
ylabel("理想幅度")
text(fp, 1.05, "fp=1500")
text(fs, 0.1, "fs=1800")
grid(true)
```

17.编写长度为 5 的中值滤波器程序。原始未受干扰的序列为：$s[n] = 3[n(0.8)^n]$, 加性噪声信号 d[n] 为随机序列, 幅度 0.6, 分别绘制长度为 40 的受干扰序列, 以及中值滤波器的输出。

```
using TyPlot
using Statistics # 导入统计包以使用 median 函数

# 1. 信号生成参数
N = 40              # 序列长度
n = 0:N-1           # 时间索引 (0 到 39)

# 2. 生成原始信号 s[n]
# 公式: s[n] = 3 * n * (0.8)^n
# 注意: Julia 中 .^ 用于逐元素幂运算
s = 3 .* n .* (0.8).^n

# 3. 生成加性噪声 d[n]
# "幅度 0.6" 通常指均匀分布在 [-0.6, 0.6] 之间的随机噪声
# rand(N) 生成 [0, 1) 的随机数
# (rand(N) .- 0.5) .* 2 生成 [-1, 1)
# 乘以 0.6 得到 [-0.6, 0.6)
noise_amp = 0.6
d = (rand(N) .- 0.5) .* 2 .* noise_amp

# 生成受干扰序列 x[n]
x = s .+ d

# 4. 实现长度为 5 的中值滤波器
# 窗口长度 L = 5，意味着取 n-2, n-1, n, n+1, n+2 的中位数
L = 5
half_L = floor(Int, L / 2) # 半窗口长度 = 2
y = zeros(N) # 初始化输出序列
```

```
for i in 1:N
    # 确定当前窗口的起始和结束索引 (处理边界情况)
    # Julia 索引从 1 开始，对应 n = i-1
    # 窗口范围: [max(1, i - half_L), min(N, i + half_L)]
    start_idx = max(1, i - half_L)
    end_idx = min(N, i + half_L)

    # 提取窗口内的数据
    window_data = x[start_idx:end_idx]

    # 计算中值并赋值
    y[i] = median(window_data)
end

# 5. 绘图
figure("Median Filter Analysis")

# 子图 1: 受干扰的序列 x[n]
subplot(2, 1, 1)
stem(n, x, "b-o", label="受干扰序列")
# 可选: 绘制原始纯净信号作为参考 (虚线)
plot(n, s, "g--", label="原始信号")
title("受干扰序列 (原始信号 + 噪声)")
ylabel("幅度")
legend()
grid(true)

# 子图 2: 中值滤波器输出 y[n]
subplot(2, 1, 2)
stem(n, y, "r-o", label="滤波后输出")
title("5点中值滤波器输出")
xlabel("样本 n")
ylabel("幅度")
legend()
grid(true)
```

18.已知 16 点序列 x[n] 的 DFT 为: $X[k] = \begin{cases} k/16 & 0 \le k \le 15 \\ 0 & otherwise \end{cases}$。绘制序列 x[n] 的实部和虚部。

```
using TyPlot
using FFTW # 需要 FFTW 包来进行快速傅里叶变换

# 1. 定义参数和频域序列 X[k]
N = 16
k = 0:N-1

# 根据题目定义: X[k] = k/16, for 0 <= k <= 15
X_k = k ./ 16.0

# 注意: 这里的 X[k] 是纯实数且不满足共轭对称性 (即 X[k] != conj(X[N-k]))
# 因此，反变换后的 x[n] 将是复数。

# 2. 计算 IDFT 得到 x[n]
# ifft 计算公式为: x[n] = (1/N) * sum(X[k] * exp(j*2*pi*n*k/N))
x_n = ifft(X_k)

# 3. 提取实部和虚部
x_real = real(x_n)
x_imag = imag(x_n)

# 打印部分数值结果供参考
println("x[n] 的前 5 个值 (复数形式):")
for i in 1:5
    println("n=$(i-1): $(round(x_n[i], digits=4))")
end

# 4. 绘图
figure("Sequence x[n] Real and Imaginary Parts")

# 子图 1: 实部
subplot(2, 1, 1)
stem(k, x_real, "b-o", label="实部")
title("x[n] 的实部")
ylabel("Re{x[n]}")
grid(true)
legend()
```

```
# 子图 2: 虚部
subplot(2, 1, 2)
stem(k, x_imag, "r-d", label="虚部") # 使用不同的颜色和标记
title("x[n] 的虚部")
xlabel("n")
ylabel("Im{x[n]}")
grid(true)
legend()
```

19.已知系统的系统函数为: $H(z) = \frac{1-0.2z^{-1}+0.5z^{-2}}{1+3.2z^{-1}+1.5z^{-2}+0.8z^{-3}+1.4z^{-4}}$ 。用 MATLAB 测试该系统的稳定性。

```
using TyPlot
using Polynomials # 用于求多项式的根

# 1. 定义系统参数
# H(z) = (1 - 0.2z^-1 + 0.5z^-2) / (1 + 3.2z^-1 + 1.5z^-2 + 0.8z^-3 + 1.4z^-4)

# 将分子分母同时乘以 z^4 转换为 z 的正幂次多项式，以便求根
# Numerator(z)   = z^4 - 0.2z^3 + 0.5z^2 = 0*z^0 + 0*z^1 + 0.5*z^2 - 0.2*z^3 + 1*z^4
# Denominator(z) = z^4 + 3.2z^3 + 1.5z^2 + 0.8z + 1.4 = 1.4*z^0 + 0.8*z^1 + 1.5*z^2 + 3.2*z^3 + 1*z^4

# Polynomials 包的系数顺序通常为升幂排列 [a0, a1, ..., an] 对应 a0 + a1*x + ...
# 分子系数 (升幂)
b_coeffs_asc = [0.0, 0.0, 0.5, -0.2, 1.0]
# 分母系数 (升幂)
a_coeffs_asc = [1.4, 0.8, 1.5, 3.2, 1.0]

# 2. 计算零点和极点
# 构造多项式对象
num_poly = Polynomial(b_coeffs_asc)
den_poly = Polynomial(a_coeffs_asc)

# 求根
zeros_z = Polynomials.roots(num_poly)
poles_z = Polynomials.roots(den_poly)

# 3. 稳定性判断
# 计算极点模长
poles_abs = abs.(poles_z)

# 判断是否所有极点都在单位圆内
is_stable = all(poles_abs .< 1.0)

# 输出结果
println("---------- 系统稳定性分析 ----------")
println("系统极点 (Poles):")
for p in poles_z
    println("  $p (模长: $(round(abs(p), digits=4)))")
end

println("\n最大极点模长: $(round(maximum(poles_abs), digits=4))")

if is_stable
    println("结论: 系统是稳定的 (所有极点在单位圆内)。")
else
    println("结论: 系统是不稳定的 (存在极点在单位圆外)。")
end
println("-----------------------------------")

# 4. 绘制零极点图 (Pole-Zero Plot)
figure("Pole-Zero Plot")
hold("on") # 修正：使用 "on" 字符串而不是布尔值

# 绘制单位圆
theta = 0:0.01:2*pi
unit_circle_x = cos.(theta)
unit_circle_y = sin.(theta)
plot(unit_circle_x, unit_circle_y, "k--", label="单位圆")

# 绘制零点 (圆圈 o)
plot(real(zeros_z), imag(zeros_z), "bo", markersize=8, label="零点 (Zeros)")

# 绘制极点 (叉号 x)
plot(real(poles_z), imag(poles_z), "rx", markersize=8, label="极点 (Poles)")

title("系统零极点分布图")
```

```
xlabel("实部 (Real)")
ylabel("虚部 (Imaginary)")
legend()
grid(true)
axis("equal") # 保持横纵坐标比例一致，使圆看起来是圆的
```

20.利用 Remez 算法设计一个等波纹线性相位 FIR 低通滤波器。该滤波器的性能指标为: 通带截止频率为 1500Hz, 阻带截止频率为 1800Hz, 通带波纹为 $\delta_p = 0.015$, 阻带波纹为 $\delta_s = 0.021$, 抽样频率为 5000Hz。

```
using TyPlot
using DSP # 包含 remez 函数和 freqz 函数

# 1. 定义设计指标
Fs = 5000.0      # 采样频率
fp = 1500.0      # 通带截止频率
fs = 1800.0      # 阻带截止频率
dp = 0.015       # 通带波纹
ds = 0.021       # 阻带波纹

# 2. 估算滤波器阶数 (利用 Herrmann 公式)
# 归一化过渡带宽度
Delta_F = (fs - fp) / Fs
log_dp = log10(dp)
log_ds = log10(ds)

# Herrmann 公式系数
a_dp = 0.005309 * (log_dp^2) + 0.07114 * log_dp - 0.4761
g_dp = -0.00266 * (log_dp^2) - 0.5941 * log_dp - 0.4278
D_inf = a_dp * log_ds + g_dp
f_corr = 11.01217 + 0.51244 * (log_dp - log_ds)
N_exact = (D_inf - f_corr * (Delta_F^2)) / Delta_F

# 取整得到滤波器长度 N
N = ceil(Int, N_exact)
# 确保 N 为奇数（对于等波纹低通滤波器，通常建议奇数长度/偶数阶，以便获得更好的特性）
if N % 2 == 0
    N += 1
end

println("估算的滤波器长度 N: $N")

# 3. 使用 Remez 算法设计滤波器
# 权重计算: 权重与波纹成反比
weights = [1.0/dp, 1.0/ds]

# 定义频带: 必须是扁平向量 [start1, stop1, start2, stop2]
# 之前的写法 bands = [(0.0, fp), (fs, Fs/2)] 是错误的
bands = [0.0, fp, fs, Fs/2]

# 定义期望幅度: [1.0, 0.0] (通带为1，阻带为0)
desired = [1.0, 0.0]

# 调用 remez 函数
# 注意: remez(n_taps, bands, desired; weight=..., Hz=...)
h = remez(N, bands, desired, weight=weights, Hz=Fs)

# 4. 计算频率响应
# 使用 freqz 计算复数频率响应
# freqz(b, a, n_points) -> (H, w) or freqz(Filter, range, Fs)
# 这里手动构造频率向量进行计算，或者使用 DSP 的 freqz
f_range = range(0, Fs/2, length=1024)
H_resp = freqz(PolynomialRatio(h, [1.0]), f_range, Fs)

# 计算幅度 (dB)
mag_db = 20 .* log10.(abs.(H_resp))

# 5. 绘图
figure("Remez Filter Design")

# 子图1: 幅频响应 (dB)
subplot(2, 1, 1)
plot(f_range, mag_db, "b-", linewidth=1.5)
title("FIR 低通滤波器幅频响应 (Remez 算法, N=$N)")
ylabel("幅度 (dB)")
```

```
xlabel("频率 (Hz)")
grid(true)

# 添加规格限制线以便观察
hold("on")
# 通带下限 (20log10(1-dp)) 和上限 (20log10(1+dp))
pass_lower = 20 * log10(1 - dp)
pass_upper = 20 * log10(1 + dp)
# 阻带上限 (20log10(ds))
stop_limit = 20 * log10(ds)

plot([0, fp], [pass_lower, pass_lower], "g--")
plot([0, fp], [pass_upper, pass_upper], "g--")
plot([fs, Fs/2], [stop_limit, stop_limit], "r--")
text(fp, pass_lower-5, "通带波纹限制")
text(fs, stop_limit+5, "阻带衰减限制")

# 限制 Y 轴范围以便更好地观察细节
ylim(-80, 5)

# 子图2: 冲激响应 h[n]
subplot(2, 1, 2)
stem(0:N-1, h, "k-o", markersize=4)
title("滤波器冲激响应 h[n]")
xlabel("n")
ylabel("幅度")
grid(true)
```

21. 已知序列 $x_1[n] = \{2.2, 3, -1.5, 4.2, -1.8\}, x_2[n] = \{0.8, -1, 1.6, 0.8\}$, 计算两个序列的卷积 $x[n] = x_1[n] * x_2[n]$, 并绘制序列 x[n]。

```
# 导入必要的包
using TyPlot
using DSP  # 用于 conv 函数。如果未安装，需先运行 import Pkg; Pkg.add("DSP")

# 1. 定义序列 x1[n] 和 x2[n]
x1 = [2.2, 3.0, -1.5, 4.2, -1.8]
x2 = [0.8, -1.0, 1.6, 0.8]

# 2. 计算卷积 x[n] = x1[n] * x2[n]
# DSP 包中的 conv 函数可以直接计算线性卷积
# 注意：由于 TyMath 和 DSP 都导出了 conv，为了避免冲突，必须显式指定 DSP.conv
xn = DSP.conv(x1, x2)

# 3. 确定时间轴 n
# 假设 x1 和 x2 均从 n=0 开始
# 卷积结果的长度为 L1 + L2 - 1
L = length(xn)
n = 0:(L - 1)

# 4. 打印计算结果
println("卷积结果 x[n]:")
println(xn)

# 5. 使用 TyPlot 绘图
# 初始化图形窗口
figure("Convolution Analysis")

# 使用 stem 绘制离散序列 (火柴杆图)
# 移除了导致报错的 "filled" 参数
stem(n, xn)

# 添加图形装饰
title("序列卷积结果 x[n] = x1[n] * x2[n]")
xlabel("n (样本点)")
ylabel("x[n] 幅度")

# 打开网格
grid("on")

# 保持图形显示 (在某些非交互式环境中可能需要)
# show()
```

22.已知序列 x[n] 为 $x[n]=\cos(\pi n/2), 0 \le n \le 15$, 绘制序列 x[n] 的 DFT 和 DTFT 的幅度。

```
# 导入必要的包
using TyPlot
using FFTW # 用于 fft, fftshift。如果未安装，请运行 import Pkg; Pkg.add("FFTW")

# 1. 定义序列 x[n]
# 范围: 0 <= n <= 15
n = 0:15
# x[n] = cos(pi * n / 2)
# 注意使用 . 进行向量化运算
x = cos.(pi .* n ./ 2)

# 2. 计算 DFT (N点离散傅里叶变换)
# 直接对 16 点序列进行 FFT
X_k = fft(x)
# 取幅度
mag_X_k = abs.(X_k)
# 定义 DFT 的频率索引轴 k
k = 0:length(x)-1

# 3. 计算 DTFT (离散时间傅里叶变换)
# DTFT 是连续频谱。为了在计算机中绘制，我们通常通过对序列进行大量补零
# 然后做 FFT 来获得高密度的频谱样本作为近似。
K = 1024 # 定义高分辨率的点数
# 构造补零后的序列: 原始 x 加上 (K - N) 个零
x_padded = [x; zeros(K - length(x))]

# 计算补零后的 FFT
X_dtft = fft(x_padded)

# 使用 fftshift 将零频率分量移到频谱中心
X_dtft_shifted = fftshift(X_dtft)

# 取幅度
mag_X_dtft = abs.(X_dtft_shifted)

# 定义 DTFT 的频率轴 omega，范围从 -pi 到 pi
w = range(-pi, stop=pi, length=K)

# 4. 绘图
figure("DFT and DTFT Analysis")

# 子图 1: 原始序列 x[n]
subplot(3, 1, 1)
stem(n, x)
title("原始序列 x[n] = cos(\\pi n / 2)")
xlabel("n")
ylabel("x[n]")
grid("on")

# 子图 2: DFT 幅度谱 |X[k]|
subplot(3, 1, 2)
stem(k, mag_X_k)
title("DFT 幅度谱 |X[k]| (N=16)")
xlabel("k (频率索引)")
ylabel("|X[k]|")
grid("on")

# 子图 3: DTFT 幅度谱 |X(e^{j\\omega})|
subplot(3, 1, 3)
plot(w, mag_X_dtft)
title("DTFT 幅度谱 (近似)")
xlabel("\\omega (弧度/样本)")
ylabel("|X(e^{j\\omega})|")
# 设置 x 轴范围为 -pi 到 pi
xlim([-pi, pi])
grid("on")

# 调整布局以防止重叠 (如果支持)
# tight_layout()
```

23.已知 FIR 滤波器的系统函数为:

$$H(z) = 2.4 + 3.2z^{-1} + 1.5z^{-2} + 0.8z^{-3} + 1.4z^{-4} + 3.6z^{-5} + 5.2z^{-}$$

用 MATLAB 将系统函数分解为二次多项式之积, 并写出各二次多项式的表达式。

```
using Polynomials # 用于多项式求根
using Printf

# 1. 定义 FIR 滤波器系数
# H(z) = 2.4 + 3.2z^-1 + 1.5z^-2 + 0.8z^-3 + 1.4z^-4 + 3.6z^-5 + 5.2z^-6
# 对应系数向量 b
b = [2.4, 3.2, 1.5, 0.8, 1.4, 3.6, 5.2]

# 2. 求系统函数的零点
# 构建多项式 P(z) = 2.4*z^6 + ... + 5.2
# 需要反转 b 的顺序传入 Polynomial (因为 Polynomials 系数是从低次到高次)
poly_coeffs = reverse(b)
P = Polynomial(poly_coeffs)
# 使用 Polynomials.roots 求根
zeros_z = Polynomials.roots(P)

# 3. 手动分解为二阶节 (SOS)
# 策略：分离实根和复根，复根按共轭配对，实根两两配对

# 设置容差判断虚部
tol = 1e-5
complex_roots = zeros_z[abs.(imag.(zeros_z)) .> tol]
real_roots = zeros_z[abs.(imag.(zeros_z)) .<= tol]

# 对复根按实部排序，确保共轭对相邻
sort!(complex_roots, by=real)
# 对实根排序
sort!(real_roots)

# 存储配对结果
pairs = []

# 配对复根
for i in 1:2:length(complex_roots)
    if i+1 <= length(complex_roots)
        push!(pairs, (complex_roots[i], complex_roots[i+1]))
    end
end

# 配对实根
for i in 1:2:length(real_roots)
    if i+1 <= length(real_roots)
        push!(pairs, (real_roots[i], real_roots[i+1]))
    end
end

# 4. 打印结果
println("系统函数 H(z) 分解为 3 个二次多项式之积：")
# H(z) = k * H1(z) * H2(z) * H3(z)
# 我们将总增益 k = b[1] = 2.4 分配给第一节
k_total = b[1]

println("形式: H(z) = H1(z) * H2(z) * H3(z)")
println("-"^50)

for (i, pair) in enumerate(pairs)
    r1, r2 = pair

    # 计算二阶节系数
    # Section = (1 - r1*z^-1) * (1 - r2*z^-1)
    #         = 1 - (r1+r2)z^-1 + (r1*r2)z^-2
    # 对应的系数为:
    # Constant: 1
    # z^-1: -(r1 + r2)
    # z^-2: r1 * r2

    coeff_z1 = -real(r1 + r2)
    coeff_z2 = real(r1 * r2)
```

```
    # 仅对第一节应用增益 k
    gain = (i == 1) ? k_total : 1.0

    c0 = 1.0 * gain
    c1 = coeff_z1 * gain
    c2 = coeff_z2 * gain

    # 格式化符号
    sign1 = c1 >= 0 ? "+" : "-"
    sign2 = c2 >= 0 ? "+" : "-"

    @printf("H%d(z) = %.4f %s %.4fz^-1 %s %.4fz^-2\n",
            i, c0, sign1, abs(c1), sign2, abs(c2))
end

println("-"^50)
println("验证提示：各节系数相乘（卷积）应近似等于原系数 [2.4, 3.2, ...]")
```

24.已知 FIR 数字低通滤波器的性能指标为: 通带截止频率 0.35π, 阻带截止频率 0.45π, 通带和阻带波纹 $\delta = 0.01$。设计满足该滤波器的 Kaiser's 窗函数, 绘制出 Kaiser's 窗函数的增益响应。

```
using TyPlot
using DSP
using FFTW

# 1. 滤波器性能指标
wp = 0.35 * pi  # 通带截止频率
ws = 0.45 * pi  # 阻带截止频率
delta = 0.01    # 波纹 (通带和阻带相同)

# 2. 计算 Kaiser 窗参数
# 2.1 计算最小阻带衰减 A (dB)
A = -20 * log10(delta)
println("目标衰减量 A = $A dB")

# 2.2 计算过渡带宽度 dw (rad)
dw = ws - wp
println("过渡带宽度 dw = $(dw/pi) * pi")

# 2.3 估算滤波器阶数 N (经验公式)
# 公式: N >= (A - 8) / (2.285 * dw)
N_float = (A - 8) / (2.285 * dw)
N = ceil(Int, N_float)
# 确保 N 为偶数 (通常 Type I FIR 滤波器阶数为偶数，长度为奇数)
if N % 2 != 0
    N += 1
end
L = N + 1 # 窗函数长度
println("估算滤波器阶数 N = $N (窗长 L = $L)")

# 2.4 计算形状参数 beta
# 公式:
# if A > 50: beta = 0.1102(A - 8.7)
# if 21 <= A <= 50: beta = 0.5842(A - 21)^0.4 + 0.07886(A - 21)
# if A < 21: beta = 0
if A > 50
    beta = 0.1102 * (A - 8.7)
elseif A >= 21
    beta = 0.5842 * (A - 21)^0.4 + 0.07886 * (A - 21)
else
    beta = 0.0
end
println("Kaiser 窗参数 beta = $beta")

# 3. 生成 Kaiser 窗函数 w[n]
# DSP 包中的 kaiser 函数生成长度为 L 的窗
w = kaiser(L, beta)

# 4. 计算窗函数的频率响应 (增益响应)
# 为了获得光滑的频谱曲线，进行大量补零 (例如补到 1024 点)
nfft = 1024

# 修复：Julia 的 fft 函数第二个参数是变换维度，不是 FFT 长度。
# 若要进行 N 点 FFT（补零），必须手动先对数组进行补零。
w_padded = [w; zeros(nfft - length(w))]
```

```
# 对补零后的序列进行 FFT
W = fft(w_padded)
W_shifted = fftshift(W) # 将零频移到中心
mag_W = abs.(W_shifted)

# 归一化幅度 (最大值为 0 dB)
# 注意：对于窗函数本身的频谱，通常看其主瓣宽度和旁瓣衰减
# 这里我们将直流分量归一化到 0dB 以便观察相对衰减
mag_W_db = 20 * log10.(mag_W ./ maximum(mag_W))

# 频率轴 (-pi 到 pi)
freq_axis = range(-pi, pi, length=nfft)

# 5. 绘图
figure("Kaiser Window Design")

# 子图1: 时域波形 w[n]
subplot(2, 1, 1)
n_seq = 0:(L-1)
stem(n_seq, w)
title("Kaiser 窗函数时域波形 (N=$N, \\beta=$(round(beta, digits=2)))")
xlabel("n")
ylabel("幅度")
grid("on")

# 子图2: 频域增益响应 (dB)
subplot(2, 1, 2)
plot(freq_axis ./ pi, mag_W_db)
title("Kaiser 窗函数幅频响应 (增益)")
xlabel("归一化频率 (\\times \\pi rad/sample)")
ylabel("幅度 (dB)")
xlim([-1, 1])
ylim([-100, 10]) # 限制 Y 轴范围以便观察旁瓣
grid("on")

# 标记主瓣和旁瓣衰减的辅助线 (可选)
# 理论旁瓣峰值应该在 -A dB 左右 (相对于归一化后的主瓣其实是衡量相对值，
# 但对于窗函数谱，第一旁瓣通常由 beta 决定)
```

25. 已知系统的频响特性为：$H(e^{j\omega}) = \frac{1-0.2e^{-j\omega}+0.5e^{-2j\omega}+2e^{-3j\omega}-0.6e^{-4j\omega}}{1+3.2e^{-j\omega}+1.5e^{-2j\omega}-0.8e^{-3j\omega}+1.4e^{-4j\omega}}$ 绘制该系统的幅频特性和相频特性。

```
using TyPlot
using FFTW
using DSP

# 1. 定义系统参数
# H(e^jw) = B(e^jw) / A(e^jw)
# 分子系数 (对应 e^0, e^-jw, e^-2jw...)
b = [1.0, -0.2, 0.5, 2.0, -0.6]
# 分母系数
a = [1.0, 3.2, 1.5, -0.8, 1.4]

# 2. 计算频率响应
# 设置 FFT 点数（分辨率），点数越多曲线越平滑
N = 1024

# 对系数向量进行补零，使其长度达到 N
# 这是为了利用 FFT 计算离散时间傅里叶变换 (DTFT) 的近似样本
b_padded = [b; zeros(N - length(b))]
a_padded = [a; zeros(N - length(a))]

# 分别计算分子和分母的 FFT
# FFT 计算结果对应频率范围 0 到 2pi (单位圆一圈)
H_num = fft(b_padded)
H_den = fft(a_padded)

# 系统频率响应 H(w) = Num(w) / Den(w)
H_w = H_num ./ H_den

# 3. 提取 0 到 pi 的部分 (通常只需分析前半部分)
half_idx = 1:(div(N, 2) + 1)
H_half = H_w[half_idx]

# 定义对应的频率轴 (0 到 pi)
w_axis = range(0, pi, length=length(half_idx))
```

```
# 4. 计算幅度响应和相位响应
# 幅度响应 (dB)
mag_response = 20 * log10.(abs.(H_half))

# 相位响应 (弧度)
phase_response = angle.(H_half)

# 手动实现相位解卷绕 (unwrap) 以获得连续曲线
# 替代 DSP.unwrap 以避免环境依赖和作用域警告
function manual_unwrap(p)
    n = length(p)
    result = copy(p)
    correction = 0.0
    for i in 2:n
        diff = p[i] - p[i-1]
        # 如果相邻点相位跳变超过 pi，则调整 2*pi
        if diff > pi
            correction -= 2 * pi
        elseif diff < -pi
            correction += 2 * pi
        end
        result[i] += correction
    end
    return result
end

phase_response = manual_unwrap(phase_response)

# 5. 绘图
figure("System Frequency Response Analysis")

# 子图 1: 幅频特性
subplot(2, 1, 1)
plot(w_axis ./ pi, mag_response)
title("幅频特性 (Magnitude Response)")
xlabel("归一化频率 (\\times \\pi rad/sample)")
ylabel("幅度 (dB)")
grid("on")

# 子图 2: 相频特性
subplot(2, 1, 2)
plot(w_axis ./ pi, phase_response)
title("相频特性 (Phase Response)")
xlabel("归一化频率 (\\times \\pi rad/sample)")
ylabel("相位 (radians)")
grid("on")

# 调整布局
# tight_layout()
```

26. 已知序列 $x_1[n]=\{2.2,3,-1.5,4.2,-1.8\}$，$x_2[n]=\{0.8,-1,1.6,0.8\}$，基于 DFT 计算两个序列的卷积 $x[n]=x_1[n]*x_2[n]$，并绘制基于 DFT 计算得到的 x[n]。

```
using TyPlot
using FFTW # 必须引入 FFTW 库来计算 DFT/IDFT

# 1. 定义已知序列
x1 = [2.2, 3.0, -1.5, 4.2, -1.8]
x2 = [0.8, -1.0, 1.6, 0.8]

# 2. 确定卷积后的长度 L
# 线性卷积的长度 = N1 + N2 - 1
N1 = length(x1)
N2 = length(x2)
L = N1 + N2 - 1

println("序列 x1 长度: $N1")
println("序列 x2 长度: $N2")
println("线性卷积所需最小 DFT 长度 L: $L")

# 3. 对序列进行补零 (Zero Padding)
# 将序列延长到长度 L，不足的部分补 0
x1_padded = [x1; zeros(L - N1)]
x2_padded = [x2; zeros(L - N2)]
```

```
# 4. 执行 DFT (FFT 算法)
X1_k = fft(x1_padded)
X2_k = fft(x2_padded)

# 5. 在频域相乘
# 卷积定理: 时域卷积 <-> 频域乘积
X_k = X1_k .* X2_k

# 6. 执行 IDFT (逆变换) 并取实部
# ifft 得到的结果通常包含极小的虚部误差，需要取 real()
x_conv = real(ifft(X_k))

# 7. 打印结果
println("基于 DFT 计算的卷积结果 x[n]:")
println(x_conv)

# 8. 绘图
figure("Convolution via DFT")

# 定义时间轴 n
n = 0:(L-1)

# 使用 stem 绘制离散序列
stem(n, x_conv)

title("基于 DFT 计算的线性卷积 x[n]")
xlabel("n (样本)")
ylabel("幅度")
grid("on")
```

27. 已知 IIR 滤波器的系统函数为：$H(z)=\frac{2+5z^{-1}+z^{-2}-3z^{-3}+4z^{-4}+6z^{-5}}{1+3z^{-1}-5z^{-2}+2z^{-3}-4z^{-4}+3z^{-5}}$。用 MATLAB 将系统函数表示为级联型结构形式，并写出各级联子系统的表达式。

```
using Polynomials
using Printf

# 1. 定义系统函数系数
# H(z) = B(z) / A(z)
# 分子系数 b: 2 + 5z^-1 + 1z^-2 - 3z^-3 + 4z^-4 + 6z^-5
b = [2.0, 5.0, 1.0, -3.0, 4.0, 6.0]
# 分母系数 a: 1 + 3z^-1 - 5z^-2 + 2z^-3 - 4z^-4 + 3z^-5
a = [1.0, 3.0, -5.0, 2.0, -4.0, 3.0]

println("原系统函数 H(z):")
println("分子 b: ", b)
println("分母 a: ", a)
println("-"^60)

# 2. 求零点和极点
# 为了求 z 的根，我们需要构建多项式。
# H(z) 的各项是 z^-k，乘以 z^N 后变成关于 z 的多项式。
# 系数向量需要反转（因为 Polynomials 默认系数顺序是 [z^0, z^1, ..., z^N]）
zeros_z = Polynomials.roots(Polynomial(reverse(b)))
poles_p = Polynomials.roots(Polynomial(reverse(a)))

# 3. 零极点配对 (Pairing) 算法
# 目标：将零点和极点分组，每组包含 2 个零点和 2 个极点（不足补0），形成二阶节
function pair_roots_complex(r_list)
    # 分离实根和复根
    tol = 1e-5
    complex_r = r_list[abs.(imag.(r_list)) .> tol]
    real_r = r_list[abs.(imag.(r_list)) .<= tol]

    # 简单的排序策略：按实部排序
    sort!(complex_r, by=real)
    # 修改：Julia 无法直接比较复数大小，即使它们近似为实数。
    # 必须显式指定按实部 (by=real) 进行排序。
    sort!(real_r, by=real)

    pairs = []

    # 配对复根 (共轭对)
    i = 1
```

```
    while i < length(complex_r)
        push!(pairs, (complex_r[i], complex_r[i+1]))
        i += 2
    end
    # 如果剩下一个复根 (理论上实系数多项式不会发生，除非数值误差)，暂且放入实根处理
    if i == length(complex_r)
        push!(real_r, complex_r[end])
    end

    # 配对实根
    j = 1
    while j < length(real_r)
        push!(pairs, (real_r[j], real_r[j+1]))
        j += 2
    end

    # 如果还剩一个实根 (奇数阶情况)
    if j == length(real_r)
        # 配对一个 0 (在 z 平面原点，意味着没有该项)
        # 注意：这里我们只存储根。之后转换系数时，单个根 r 对应 (z-r)
        push!(pairs, (real_r[end], nothing))
    end

    return pairs
end

zero_pairs = pair_roots_complex(zeros_z)
pole_pairs = pair_roots_complex(poles_p)

# 确保零点对和极点对数量一致 (不足的补 1，即 (z-0)(z-0) 对应的系数是 1)
# 这里假设分子分母阶数相同或相近
n_sections = max(length(zero_pairs), length(pole_pairs))

# 4. 构建级联结构 (SOS) 并输出
# 计算总增益 k = b[0] / a[0]
k_gain = b[1] / a[1]

println("级联结构分解结果 (H(z) = H1(z) * H2(z) * ...):")
println("注意: 第一个子系统包含了总增益 k = $(k_gain)")
println("-"^60)

for i in 1:n_sections
    # 获取零点对
    z_pair = i <= length(zero_pairs) ? zero_pairs[i] : (0.0, 0.0)
    # 获取极点对
    p_pair = i <= length(pole_pairs) ? pole_pairs[i] : (0.0, 0.0)

    # --- 计算分子系数 (Numerator) ---
    # (1 - z1*z^-1)(1 - z2*z^-1) = 1 - (z1+z2)z^-1 + (z1*z2)z^-2
    if z_pair[2] === nothing
        # 单实根情况: (1 - z1*z^-1)
        z1 = z_pair[1]
        b_sec = [1.0, -real(z1), 0.0]
    else
        z1, z2 = z_pair
        b_sec = [1.0, -real(z1 + z2), real(z1 * z2)]
    end

    # --- 计算分母系数 (Denominator) ---
    if p_pair[2] === nothing
        p1 = p_pair[1]
        a_sec = [1.0, -real(p1), 0.0]
    else
        p1, p2 = p_pair
        a_sec = [1.0, -real(p1 + p2), real(p1 * p2)]
    end

    # --- 分配增益 ---
    if i == 1
        b_sec = b_sec .* k_gain
    end

    # --- 格式化输出 ---
    # 辅助函数：格式化项
    function fmt_term(val, power)
        if abs(val) < 1e-6 return "" end # 忽略极小值
        sign_str = val >= 0 ? "+" : "-"
        val_abs = abs(val)
        z_str = power == 0 ? "" : "z^-$power"
        return " $sign_str $(@sprintf("%.4f", val_abs))$z_str"
    end
```

```
    # 拼接字符串
    num_str = @sprintf("%.4f", b_sec[1]) * fmt_term(b_sec[2], 1) * fmt_term(b_sec[3], 2)
    den_str = @sprintf("%.4f", a_sec[1]) * fmt_term(a_sec[2], 1) * fmt_term(a_sec[3], 2)

    println("子系统 H$i(z):")
    println("      $num_str")
    println("  ----------------------------")
    println("      $den_str")
    println("")
end
```

28.用 Kaiser's 窗函数设计 FIR 数字高通滤波器，其滤波器的性能指标为：通带截止频率 0.55π，阻带截止频率 0.45π，通带和阻带波纹 $\delta = 0.04$。绘制出该滤波器的增益响应。

```
using TyPlot
using DSP
using FFTW
# 移除 SpecialFunctions 引用以避免命名冲突

# 1. 滤波器性能指标
# 高通滤波器
ws = 0.45 * pi  # 阻带截止频率
wp = 0.55 * pi  # 通带截止频率
delta = 0.04    # 波纹 (通带和阻带相同)

# 2. 计算参数
# 截止频率 wc (取中心)
wc = (ws + wp) / 2

# 过渡带宽度 dw
dw = abs(wp - ws)

# 阻带衰减 A (dB)
A = -20 * log10(delta)
println("设计参数:")
println("  目标衰减 A = $(round(A, digits=2)) dB")
println("  截止频率 wc = $(wc/pi) * pi")
println("  过渡带 dw = $(dw/pi) * pi")

# 估算阶数 N (使用 Kaiser 经验公式)
# N = (A - 8) / (2.285 * dw)
N_est = (A - 8) / (2.285 * dw)
N = ceil(Int, N_est)
# 保证 N 为偶数 (滤波器长度 M 为奇数)，以保证 Type I FIR (对称，中心不为0)
# 高通滤波器通常需要 Type I (奇数长度 symmetric) 或 Type IV (偶数长度 antisymmetric)
# 这里选择奇数长度 (Type I)
if N % 2 != 0
    N += 1
end
M = N + 1 # 滤波器长度 (Tap 数)
println("  滤波器阶数 N = $N (长度 M = $M)")

# 计算 beta 参数
# 将变量名 beta 修改为 beta_val 避免与函数名冲突
if A > 50
    beta_val = 0.1102 * (A - 8.7)
elseif A >= 21
    beta_val = 0.5842 * (A - 21)^0.4 + 0.07886 * (A - 21)
else
    beta_val = 0.0
end
println("  Kaiser 参数 beta = $(round(beta_val, digits=4))")

# 3. 构造理想高通滤波器 h_d[n]
# 理想高通 = 延迟脉冲 - 理想低通
# h_hp[n] = delta[n - tau] - (wc/pi) * sinc((wc/pi) * (n - tau))
tau = (M - 1) / 2
n = 0:(M-1)

# 注意：Julia 的 sinc(x) 定义为 sin(pi*x)/(pi*x)
# 公式中的 sinc 参数需要归一化
# 理想低通部分系数
h_lp = (wc / pi) .* sinc.((wc / pi) .* (n .- tau))
```

```
# 理想高通系数
h_d = -h_lp
# 在中心点 (n = tau) 加上 1 (即 delta 函数)
# 注意索引，Julia 是 1-based，但在 n 向量中我们已经定义了 0 到 M-1
# tau 对应的索引是 tau + 1
center_idx = Int(tau) + 1
h_d[center_idx] += 1.0

# 4. 生成 Kaiser 窗并加窗
w = kaiser(M, beta_val)
h = h_d .* w

# 5. 计算增益响应
nfft = 1024
# 补零
h_padded = [h; zeros(nfft - length(h))]
H = fft(h_padded)
# 取前半部分 (0 到 pi) 进行绘制，这样更直观
half_len = div(nfft, 2) + 1
H_half = H[1:half_len]
# 计算幅度 (dB)
mag_H_db = 20 * log10.(abs.(H_half))

# 频率轴 (0 到 pi)
w_axis = range(0, pi, length=half_len)

# 6. 绘图
figure("High-pass Filter Design using Kaiser Window")

# 子图1: 滤波器系数 h[n]
subplot(2, 1, 1)
stem(n, h)
title("FIR 高通滤波器单位脉冲响应 h[n] (N=$N)")
xlabel("n")
ylabel("幅度")
grid("on")

# 子图2: 增益响应 (dB)
subplot(2, 1, 2)
plot(w_axis ./ pi, mag_H_db)
title("增益响应 (Gain Response) - [0, \\pi]")
xlabel("归一化频率 (\\times \\pi rad/sample)")
ylabel("幅度 (dB)")
xlim([0, 1])    # 显示 0 到 1 (pi)
ylim([-80, 10]) # 限制 Y 轴范围
grid("on")

# 添加指标辅助线
# 阻带截止 (0.45) 和 通带截止 (0.55)
# 阻带上限 (-A dB)
plot([0, 0.45], [-A, -A], "--r", label="阻带指标")
# 通带下限 (这里近似画在 0dB 附近示意)
plot([0.55, 1], [0, 0], "--g", label="通带指标")
legend()
```

29.绘制 6 点滑动平均滤波器的幅频特性和相频特性。

```
using TyPlot
using FFTW

# 1. 定义滤波器参数
# 6 点滑动平均滤波器
# 系统函数 H(z) = (1/6) * (1 + z^-1 + ... + z^-5)
# 滤波器系数 (单位脉冲响应 h[n])
M = 6
h = ones(M) ./ M

# 2. 计算频率响应
# 使用 FFT 计算离散时间傅里叶变换 (DTFT) 的近似
# 补零到较长的长度以获得平滑的曲线
nfft = 1024
# 补零
h_padded = [h; zeros(nfft - length(h))]

# 计算 FFT
H = fft(h_padded)
```

```
# 3. 提取 0 到 pi 的部分 (单边频谱)
# 因为 h[n] 是实数，频谱是共轭对称的，只需分析前半部分
half_len = div(nfft, 2) + 1
H_half = H[1:half_len]

# 定义归一化频率轴 (0 到 1, 对应 0 到 pi)
w_axis = range(0, 1, length=half_len)

# 4. 计算幅频特性和相频特性
# 幅度 (dB)
mag_response = 20 * log10.(abs.(H_half))

# 相位 (弧度)
phase_response = angle.(H_half)

# 手动实现相位解卷绕 (Unwrap)
# 滑动平均滤波器是线性相位滤波器，理论上相位是一条直线
# 但 angle() 函数的结果限制在 [-pi, pi]，会导致跳变
function manual_unwrap(p)
    n = length(p)
    result = copy(p)
    correction = 0.0
    for i in 2:n
        diff = p[i] - p[i-1]
        if diff > pi
            correction -= 2 * pi
        elseif diff < -pi
            correction += 2 * pi
        end
        result[i] += correction
    end
    return result
end

phase_unwrapped = manual_unwrap(phase_response)

# 5. 绘图
figure("6-Point Moving Average Frequency Response")

# 子图 1: 幅频特性
subplot(2, 1, 1)
plot(w_axis, mag_response)
title("6点滑动平均滤波器 - 幅频特性")
xlabel("归一化频率 (\\times \\pi rad/sample)")
ylabel("幅度 (dB)")
grid("on")
# 限制 y 轴范围以避免 -Inf 对显示的影响 (在零点处)
ylim([-50, 5])

# 子图 2: 相频特性
subplot(2, 1, 2)
plot(w_axis, phase_unwrapped)
title("6点滑动平均滤波器 - 相频特性")
xlabel("归一化频率 (\\times \\pi rad/sample)")
ylabel("相位 (radians)")
grid("on")

# 验证线性相位特性：相位应该是一条斜率为 -(M-1)/2 = -2.5 的直线
# 对于 w (0~pi), slope = -2.5.
# 归一化频率 x = w/pi.
# 所以 plot(x, phase) 的斜率应该是 -2.5 * pi
```

30.原始序列为: $s[n] = 3[n(0.8)^n]$, 加性噪声 d[n] 为随机序列, 幅度 0.6, 受干扰的序列为: $x[n] = s[n] + d[n]$, 使用重叠相加法实现 5 点滑动平均滤波器对 x[n] 的处理。绘制未受干扰的序列 s[n] 和滤波器输出的有噪序列 (利用 `fftfilt` 函数)。

```
using TyPlot
using DSP
using Random

# 1. 生成信号和噪声
# 设定序列长度
N = 60
n = 0:(N-1)
```

```
# 原始序列 s[n] = 3 * n * (0.8)^n
s = 3 .* n .* (0.8).^n

# 生成加性噪声 d[n]
# "幅度 0.6" 这里理解为噪声在 [-0.6, 0.6] 之间均匀分布
Random.seed!(123) # 固定随机种子以保证结果可复现
d = 0.6 .* (2 .* rand(N) .- 1)

# 受干扰的序列 x[n]
x = s + d

# 2. 定义滤波器
# 5 点滑动平均滤波器
# h[n] = [1/5, 1/5, 1/5, 1/5, 1/5]
M = 5
h = ones(M) ./ M

# 3. 使用 fftfilt 进行滤波
# fftfilt 函数利用重叠相加法(Overlap-Add)高效计算长序列的卷积
y = fftfilt(h, x)

# 4. 绘图
figure("Overlap-Add Filtering using fftfilt")

# 子图 1: 原始未受干扰序列 s[n]
subplot(3, 1, 1)
stem(n, s)
title("原始序列 s[n] = 3n(0.8)^n")
xlabel("n")
ylabel("幅度")
grid("on")

# 子图 2: 受干扰的序列 x[n] (为了对比展示)
subplot(3, 1, 2)
stem(n, x)
title("受干扰序列 x[n] = s[n] + d[n]")
xlabel("n")
ylabel("幅度")
grid("on")

# 子图 3: 滤波器输出的有噪序列 y[n]
subplot(3, 1, 3)
stem(0:(length(y)-1), y)
title("5点滑动平均滤波器输出 (利用 fftfilt)")
xlabel("n")
ylabel("幅度")
grid("on")

# 调整布局
# tight_layout()
```

31. 已知 IIR 滤波器的系统函数为：$H(z)=\frac{2+5z^{-1}+z^{-2}-3z^{-3}+4z^{-4}+6z^{-5}}{1+3z^{-1}-5z^{-2}+2z^{-3}-4z^{-4}+3z^{-5}}$。用 MATLAB 对系统进行并联结构 I 型和并联结构 II 型分解。

```
using LinearAlgebra
using TyPlot  # 使用 Syslab 原生绘图库

# ==============================================================================
#  第一部分：核心算法 (手动实现 residuez)
# ==============================================================================

# 1. 求多项式根 (使用伴随矩阵法，数值稳定性高)
function get_roots_robust(coeffs)
    n = length(coeffs) - 1
    # 归一化，防止首项不为1
    if abs(coeffs[1]) < 1e-9; return ComplexF64[]; end
    c = coeffs ./ coeffs[1]

    # 构建伴随矩阵
    Cm = zeros(ComplexF64, n, n)
    for i in 1:n-1; Cm[i+1, i] = 1.0; end
    for i in 1:n; Cm[i, n] = -c[i+1]; end
```

```
    return eigvals(Cm)
end

# 2. 部分分式展开 (Residuez)
function my_residuez(b, a)
    # 计算直接项 k (长除法首项)
    k = 0.0
    if length(b) == length(a); k = b[1] / a[1]; end

    # 计算极点 p
    p = get_roots_robust(a)

    # 计算留数 r (使用留数定理导数法)
    r = zeros(ComplexF64, length(p))
    b_new = b .- k .* a # 去除直接项后的分子

    for i in 1:length(p)
        pi = p[i]
        # 分子值 Num(pi)
        num_val = sum(b_new[j] * (pi ^ (-(j-1))) for j in 1:length(b_new))
        # 分母导数值 Den'(pi) (等效于去除该极点后的连乘)
        den_val = 1.0 + 0.0im
        for j in 1:length(p)
            if i != j; den_val *= (1 - p[j]/pi); end
        end
        r[i] = num_val / den_val
    end
    return r, p, k
end

# ==============================================================================
#  第二部分：主程序与结果输出
# ==============================================================================

# 1. 输入题目给定的系数
# H(z) 分子: 2 + 5z^-1 + 1z^-2 - 3z^-3 + 4z^-4 + 6z^-5
b = [2.0, 5.0, 1.0, -3.0, 4.0, 6.0]
# H(z) 分母: 1 + 3z^-1 - 5z^-2 + 2z^-3 - 4z^-4 + 3z^-5
a = [1.0, 3.0, -5.0, 2.0, -4.0, 3.0]

println("正在进行 IIR 滤波器并联分解...\n")
r, p, k = my_residuez(b, a)

# 2. 打印计算结果 (作业答案)
println("============================================================")
println("                分解结果 (可直接抄写)")
println("============================================================")

# --- 输出并联结构 I 型 (复数形式) ---
println("\n[1] 并联结构 I 型 (Complex Form):")
println("    H(z) = k + Σ [ r_i / (1 - p_i z^-1) ]")
println("------------------------------------------------------------")
print("H(z) = ", round(real(k), digits=4))

for i in 1:length(p)
    # 格式化复数
    r_re = round(real(r[i]), digits=4); r_im = round(imag(r[i]), digits=4)
    r_str = abs(r_im)<1e-4 ? "$r_re" : "($r_re + $(r_im)j)"

    p_re = round(real(p[i]), digits=4); p_im = round(imag(p[i]), digits=4)
    p_str = abs(p_im)<1e-4 ? "$p_re" : "($p_re + $(p_im)j)"

    println("")
    print("       + [ $r_str ] / ( 1 - $p_str z^-1 )")
end
println("")

# --- 输出并联结构 II 型 (实数形式 - 二阶节合并) ---
println("\n\n[2] 并联结构 II 型 (Real Form):")
println("    说明: 将共轭复数极点合并为二阶节(SOS)，系数为实数")
println("------------------------------------------------------------")
print("H(z) = ", round(real(k), digits=4))

handled = falses(length(p)) # 标记已处理的极点

for i in 1:length(p)
    if handled[i]; continue; end

    # 判定实数极点
    if abs(imag(p[i])) < 1e-5
        val_r = real(r[i]); val_p = real(p[i])
        sign_p = val_p >= 0 ? "-" : "+" # 分母显示 1 - pz^-1
```

```
            println("")
            print("       + ", round(val_r, digits=4))
            print(" / ( 1 $sign_p ", round(abs(val_p), digits=4), " z^-1 )")
            handled[i] = true
        else
            # 寻找共轭对
            for j in (i+1):length(p)
                if !handled[j] && abs(real(p[i]) - real(p[j])) < 1e-5
                    # 合并公式
                    # 分子 b0 + b1*z^-1 = 2Re(r) - 2Re(r*p')z^-1
                    b0 = 2 * real(r[i])
                    b1 = -2 * real(r[i] * conj(p[i]))

                    # 分母 1 + a1*z^-1 + a2*z^-2 = 1 - 2Re(p)z^-1 + |p|^2z^-2
                    a1 = -2 * real(p[i])
                    a2 = abs(p[i])^2

                    # 格式化
                    b0_s = round(b0, digits=4)
                    b1_s = b1 >= 0 ? "+ $(round(b1,digits=4))" : "- $(round(abs(b1),digits=4))"
                    a1_s = a1 >= 0 ? "+ $(round(a1,digits=4))" : "- $(round(abs(a1),digits=4))"
                    a2_s = round(a2, digits=4)

                    println("")
                    print("       + ( $b0_s $b1_s z^-1 )")
                    print(" / ( 1 $a1_s z^-1 + $a2_s z^-2 )")

                    handled[i] = true; handled[j] = true; break
                end
            end
        end
    end
end
println("\n")

# ==============================================================================
#  第三部分：绘图 (Pole-Zero Plot)
# ==============================================================================

# 计算零点用于绘图
zeros_val = get_roots_robust(b)

figure("IIR Filter Analysis")

# 1. 绘制单位圆
theta = range(0, 2pi, length=200)
plot(cos.(theta), sin.(theta), "k--", linewidth=1, label="Unit Circle")
hold("on")

# 2. 绘制零点 (Zeros) - 蓝色圆圈
plot(real(zeros_val), imag(zeros_val), "bo", markersize=8, label="Zeros")

# 3. 绘制极点 (Poles) - 红色叉号
plot(real(p), imag(p), "rx", markersize=10, label="Poles")

# 4. 图表修饰
title("Pole-Zero Plot of IIR Filter")
xlabel("Real Axis")
ylabel("Imaginary Axis")
grid("on")
axis("equal") # 保证圆是圆的
legend("on")

hold("off")
```

32.用海明窗设计多频带 FIR 滤波器, 该滤波器满足如下条件。在频率范围 0 到 0.32π 内幅度为 0.6, 在频率范围 0.35π 到 0.65π 内幅度为 0.2, 在频率范围 0.68π 到 π 内幅度为 0.8。绘制出该滤波器的幅频特性。

```
using LinearAlgebra
using TyPlot # 使用 Syslab 原生绘图

# ============================================================================
#  第一部分：FIR 滤波器设计核心算法 (窗函数法)
# ============================================================================

function design_multiband_fir()
    # 1. 确定参数
    # 过渡带宽度 delta_w = 0.03pi
    # 海明窗估算阶数: N ≈ 6.6pi / delta_w ≈ 6.6 / 0.03 = 220
    # 我们选择长度 L = 221 (偶对称，线性相位)
    N_order = 220
    L = N_order + 1
    tau = N_order / 2  # 中心点

    # 截止频率 (取过渡带中心)
    wc1 = 0.335 * pi  # (0.32 + 0.35)/2
    wc2 = 0.665 * pi  # (0.65 + 0.68)/2

    # 2. 计算理想脉冲响应 h_d[n]
    # 理想幅频特性可以看作三个矩形频带的叠加，或者全通+低通的组合。
    # 这里我们利用线性性质分解:
    # FullBand(0.8) - LowPass_wc2(0.6) + LowPass_wc1(0.4)
    # 解释:
    #   0 ~ wc1: 0.8 - 0.6 + 0.4 = 0.6 (满足)
    #   wc1 ~ wc2: 0.8 - 0.6 + 0 = 0.2 (满足)
    #   wc2 ~ pi:  0.8 - 0 + 0 = 0.8 (满足)

    h = zeros(Float64, L)
    for i in 0:L-1
        n_shift = i - tau

        # 处理 n=0 (L'Hopital 极限)
        if abs(n_shift) < 1e-9
            # h[0] = 0.8 - 0.6*(wc2/pi) + 0.4*(wc1/pi)
            val = 0.8 - 0.6*(wc2/pi) + 0.4*(wc1/pi)
        else
            # h[n] = 0.8*delta[n] - 0.6*sin(wc2*n)/(pi*n) + 0.4*sin(wc1*n)/(pi*n)
            # 注意：全通项 0.8*delta[n] 只在 n=0 时存在，这里 n!=0，所以只有 sinc 项
            val = -0.6 * sin(wc2 * n_shift) / (pi * n_shift) +
                  0.4 * sin(wc1 * n_shift) / (pi * n_shift)
        end
        h[i+1] = val
    end

    # 3. 生成海明窗 (Hamming Window)
    # w[n] = 0.54 - 0.46 * cos(2pi*n / (L-1))
    w_win = [0.54 - 0.46 * cos(2 * pi * i / (L - 1)) for i in 0:L-1]

    # 4. 加窗得到最终系数
    h_final = h .* w_win

    return h_final
end

# 手动计算幅频响应 (DTFT)
function calc_freq_response(h, steps=1000)
    w_vals = range(0, pi, length=steps)
    mag_vals = Float64[]

    for w in w_vals
        # H(w) = sum(h[n] * exp(-j*w*n))
        H = 0.0 + 0.0im
        for i in 1:length(h)
            H += h[i] * exp(-im * w * (i-1))
        end
        push!(mag_vals, abs(H))
    end
    return w_vals, mag_vals
end
```

```
# ============================================================================
#  第二部分：主程序与绘图
# ============================================================================

println("正在设计 FIR 滤波器 (N=220)...")
h_coeff = design_multiband_fir()

println("正在计算幅频特性...")
w_axis, mag_resp = calc_freq_response(h_coeff)

println("正在绘图...")
figure("Question 32: Multiband FIR Filter")

# 绘制幅频特性
plot(w_axis/pi, mag_resp, "b-", linewidth=1.5, label="Designed Filter")

# 绘制理想的规约线（红色虚线框），方便验证
hold("on")
# 0 ~ 0.32pi: 0.6
plot([0, 0.32], [0.6, 0.6], "r--", linewidth=2, label="Ideal Spec")
# 0.35pi ~ 0.65pi: 0.2
plot([0.35, 0.65], [0.2, 0.2], "r--", linewidth=2)
# 0.68pi ~ 1.0pi: 0.8
plot([0.68, 1.0], [0.8, 0.8], "r--", linewidth=2)

title("Frequency Response of Multiband FIR Filter (Hamming Window)")
xlabel("Normalized Frequency (x pi)")
ylabel("Magnitude")
legend("on")
grid("on")
ylim([0, 1.0]) # 设置纵坐标范围方便观察

hold("off")
```

33.已知滤波器的差分方程和输入信号为:

$$y[n] = -6.76195x[n] + 13.456335x[n-1] - 6.76195x[n-2]$$

$$x[n] = [\cos(0.1n) + \cos(0.4n)]u[n]$$

绘制该系统的输入序列和输出序列的包络。

```
using TyPlot # 使用 Syslab 原生绘图库
using LinearAlgebra

# ==========================================
#  1. 基础函数定义
# ==========================================

# 手动实现 FIR 滤波函数 (卷积)
function my_fir_filter(b, x)
    N = length(x)
    M = length(b)
    y = zeros(Float64, N)

    for n in 1:N
        # y[n] = sum(b[k] * x[n-k])
        # 注意 Julia 索引从 1 开始，所以对应公式中的 n-1, n-2...
        val = 0.0
        for k in 1:M
            if n - k + 1 > 0
                val += b[k] * x[n - k + 1]
            end
        end
        y[n] = val
    end
    return y
end

# ==========================================
#  2. 系统参数与信号生成
# ==========================================
```

```
# 滤波器系数 (根据差分方程)
# y[n] = -6.76195x[n] + 13.456335x[n-1] - 6.76195x[n-2]
b = [-6.76195, 13.456335, -6.76195]

# 时间轴 (取足够长以观察包络)
# w=0.1 对应的周期是 2pi/0.1 ≈ 62.8 点
# 我们取 0 到 150 点，大约包含 2.5 个低频周期
n = 0:200

# 生成输入信号
# x[n] = [cos(0.1n) + cos(0.4n)] * u[n]
x = [cos(0.1 * i) + cos(0.4 * i) for i in n]

# ==========================================
#  3. 执行滤波
# ==========================================

y = my_fir_filter(b, x)

# ==========================================
#  4. 绘图 (输入与输出的包络)
# ==========================================

figure("Question 33: Envelope Analysis")

# --- 子图 1: 输入序列 x[n] ---
subplot(2, 1, 1)
# 为了展示"包络"效果，我们同时画出离散杆图和连续连线
stem(n, x, "b-", markersize=2, label="Discrete Samples") # 离散点
hold("on")
plot(n, x, "b--", linewidth=1, alpha=0.5, label="Envelope") # 连线表示包络趋势
title("Input Signal x[n]: cos(0.1n) + cos(0.4n)")
xlabel("n")
ylabel("Amplitude")
grid("on")
legend("on")

# --- 子图 2: 输出序列 y[n] ---
subplot(2, 1, 2)
stem(n, y, "r-", markersize=2, label="Discrete Samples")
plot(n, y, "r--", linewidth=1, alpha=0.5, label="Envelope")
title("Output Signal y[n] (0.1 rad/s component filtered out)")
xlabel("n")
ylabel("Amplitude")
grid("on")
legend("on")

hold("off")

# ==========================================
#  5. 验证分析 (打印到控制台)
# ==========================================
println("================ 分析结果 ================")
println("输入信号包含频率: 0.1 rad 和 0.4 rad")
println("观察输出图形:")
println("1. 初始阶段有短暂的瞬态响应。")
println("2. 稳态后，输出变成单一频率的正弦波。")
println("   (因为滤波器是一个陷波器，滤除了 0.1 rad 的分量)")
```

34.已知系统的系统函数为:

$$H(z) = \frac{1-0.2z^{-1}+0.5z^{-2}+2z^{-3}-0.6z^{-4}}{1+3.2z^{-1}+1.5z^{-2}-0.8z^{-3}+1.4z^{-4}}$$

绘制该系统的零极点分布图。

```
using LinearAlgebra
using TyPlot  # 使用 Syslab 原生绘图库

# ==========================================
#  1. 基础函数：求多项式根
# ==========================================
function get_roots_robust(coeffs)
    # 输入系数向量 [a0, a1, ..., an] 对应 a0 + a1*z^-1 + ...
    n = length(coeffs) - 1
    if abs(coeffs[1]) < 1e-9; return ComplexF64[]; end

    # 归一化系数
```

```
    c = coeffs ./ coeffs[1]

    # 构建伴随矩阵 (Companion Matrix)
    # 特征值即为多项式的根
    Cm = zeros(ComplexF64, n, n)
    for i in 1:n-1; Cm[i+1, i] = 1.0; end
    for i in 1:n; Cm[i, n] = -c[i+1]; end

    return eigvals(Cm)
end

# ==========================================
#  2. 定义系统参数
# ==========================================

# 分子系数 (Numerator)
# 1 - 0.2z^-1 + 0.5z^-2 + 2z^-3 - 0.6z^-4
b = [1.0, -0.2, 0.5, 2.0, -0.6]

# 分母系数 (Denominator)
# 1 + 3.2z^-1 + 1.5z^-2 - 0.8z^-3 + 1.4z^-4
a = [1.0, 3.2, 1.5, -0.8, 1.4]

# ==========================================
#  3. 计算零点与极点
# ==========================================

zeros_val = get_roots_robust(b) # 分子根 -> 零点
poles_val = get_roots_robust(a) # 分母根 -> 极点

println("计算完成。")
println("零点 (Zeros): ", round.(zeros_val, digits=4))
println("极点 (Poles): ", round.(poles_val, digits=4))

# ==========================================
#  4. 绘制零极点分布图
# ==========================================

figure("Question 34: Pole-Zero Plot")
hold("on")

# (1) 绘制单位圆 (Unit Circle) - 参考线
theta = range(0, 2pi, length=200)
plot(cos.(theta), sin.(theta), "k--", linewidth=1, label="Unit Circle")

# (2) 绘制零点 (Zeros) - 蓝色圆圈 'o'
plot(real(zeros_val), imag(zeros_val), "bo", markersize=8, label="Zeros")

# (3) 绘制极点 (Poles) - 红色叉号 'x'
plot(real(poles_val), imag(poles_val), "rx", markersize=10, label="Poles")

# (4) 图表设置
title("Pole-Zero Plot of the System")
xlabel("Real Axis")
ylabel("Imaginary Axis")
grid("on")
legend("on")

# 【关键】设置坐标轴比例相等，确保圆画出来是圆的
axis("equal")

# 如果极点范围很大，适当调整显示范围以便看清
max_val = maximum(abs.([real(poles_val); imag(poles_val); 1.5])) + 0.5
xlim([-max_val, max_val])
ylim([-max_val, max_val])

hold("off")
```

35.已知全通系统的系统函数为: $H(z)=\frac{3-4z^{-1}+2z^{-2}-5z^{-3}+3z^{-4}+z^{-5}}{1+3z^{-1}-5z^{-2}+2z^{-3}-4z^{-4}+3z^{-5}}$ 。用 MATLAB 求全通系统进行级联格型结构的乘法器系数。

```
using LinearAlgebra

# ==========================================================================
#  1. 核心算法：Levinson-Durbin Step-Down (求反射系数)
# ==========================================================================
function poly2lattice(a_coeffs)
    # 归一化 (通常 a[1] 已经是 1)
```

```
    an = a_coeffs ./ a_coeffs[1]

    # 阶数 N
    N = length(an) - 1

    # 初始化反射系数数组 k
    k = zeros(Float64, N)

    # 当前阶数的多项式系数 (从 z^0 到 z^-m)
    a_curr = copy(an)

    # 递推循环：从高阶 N 降到 1
    for m in N:-1:1
        # 1. 提取反射系数 k_m (即当前多项式的最后一项)
        k[m] = a_curr[end]

        # 如果不是最后一轮，则计算下一阶多项式
        if m > 1
            # 2. 计算分母 (1 - k^2)
            denom = 1 - k[m]^2

            # 检查稳定性/奇异性
            if abs(denom) < 1e-9
                println("警告: 系统不稳定或临界稳定，|k| >= 1")
            end

            # 3. 降阶公式: a'_{i} = (a_i - k * a_{m-i}) / (1 - k^2)
            # 注意: Julia 索引从 1 开始 (a[1] 是 z^0, a[i+1] 是 z^-i)
            a_prev = zeros(Float64, m) # 长度为 m (阶数为 m-1)
            a_prev[1] = 1.0            # 首项始终为 1

            for i in 1:(m-1)
                # a_curr[i+1] 对应 z^-i
                # a_curr[m-i+1] 对应 z^{-(m-i)}
                a_prev[i+1] = (a_curr[i+1] - k[m] * a_curr[m - i + 1]) / denom
            end

            # 更新当前多项式
            a_curr = a_prev
        end
    end

    return k
end

# ==============================================================================
#  2. 主程序：解决第 35 题
# ==============================================================================

# 题目给定的分母系数 (Denominator Coefficients)
# D(z) = 1 + 3z^-1 - 5z^-2 + 2z^-3 - 4z^-4 + 3z^-5
a_coeffs = [1.0, 3.0, -5.0, 2.0, -4.0, 3.0]

println("正在计算格型结构系数...")
k_coeffs = poly2lattice(a_coeffs)

# ==============================================================================
#  3. 结果输出
# ==============================================================================
println("\n=============================================================")
println("            第 35 题计算结果")
println("=============================================================")
println("系统分母系数: ", a_coeffs)
println("-------------------------------------------------------------")
println("级联格型结构乘法器系数 (反射系数 k):")

for i in 1:length(k_coeffs)
    println("  k[$i] = $(round(k_coeffs[i], digits=4))")
end
println("-------------------------------------------------------------")

# 简单验证: 在全通格型结构中，k_N 应该等于 a_N
println("验证: k[5] 是否等于 a[5]? -> ", isapprox(k_coeffs[end], a_coeffs[end]))
```

36.已知有限长序列为: $x[n]=\sin(25\pi n/64), 0\le n\le 63$, 求该序列的 64 点离散傅立叶变换 X[k], 绘制出 X[k] 的幅度。

```
using TyPlot      # Syslab 原生绘图
using LinearAlgebra

# ==========================================
#  1. 定义 DFT 计算函数 (防止缺少 FFTW 包)
# ==========================================
function my_dft(x)
    N = length(x)
    X = zeros(ComplexF64, N)
    # DFT 公式: X[k] = sum_{n=0}^{N-1} x[n] * exp(-j*2*pi*k*n/N)
    for k in 0:N-1
        sum_val = 0.0 + 0.0im
        for n in 0:N-1
            # 注意：Julia 数组索引从 1 开始，所以取 x[n+1]
            angle = -2 * pi * k * n / N
            sum_val += x[n+1] * exp(im * angle)
        end
        X[k+1] = sum_val
    end
    return X
end

# ==========================================
#  2. 生成信号与计算
# ==========================================
N = 64
n = 0:N-1

# 生成序列 x[n] = sin(25*pi*n / 64)
x = sin.(25 * pi * n / 64)

println("正在计算 64 点 DFT...")
X_k = my_dft(x)

# 计算幅度谱 |X[k]|
mag_X = abs.(X_k)

# ==========================================
#  3. 绘图
# ==========================================
figure("Question 36: DFT Magnitude")

# 使用 stem 图 (火柴梗图) 展示离散频谱
# 也就是画出每一根谱线的高度
k_axis = 0:N-1
stem(k_axis, mag_X, "b-", filled=true, markersize=4, label="|X[k]|")

# 图表修饰
title("Magnitude of 64-point DFT: |X[k]|")
xlabel("Frequency Index k")
ylabel("Magnitude")
grid("on")

# 标记出峰值位置的辅助线 (k=12.5 处)
# 理论中心在 12.5 和 64-12.5=51.5
hold("on")
plot([12.5, 12.5], [0, maximum(mag_X)], "r--", linewidth=1, label="True Freq (k=12.5)")
plot([51.5, 51.5], [0, maximum(mag_X)], "r--", linewidth=1)
legend("on")

hold("off")

println("计算完成。")
println("观察图形：由于频率 k=12.5 不是整数，")
println("频谱能量并未集中在单根谱线上，而是分散在 k=12 和 k=13 周围。")
```

37.设计 4 阶巴特沃兹模拟低通滤波器, 其 3-dB 截止频率为 1, 绘制滤波器的增益响应。

```
using TyPlot       # Syslab 原生绘图
using LinearAlgebra
using Printf

# ==========================================
#  1. 巴特沃兹滤波器设计算法
# ==========================================

function design_butterworth_analog(N, Wc)
    # 计算极点位置
    # 极点分布在半径为 Wc 的左半圆上
    # 角度公式: theta_k = pi * (2k + N - 1) / (2N), k = 1...N

    poles = ComplexF64[]
    for k in 1:N
        theta = pi * (2*k + N - 1) / (2*N)
        s = Wc * exp(im * theta)
        push!(poles, s)
    end

    # 根据极点构建分母多项式系数 D(s) = (s-p1)(s-p2)...
    # 我们使用多项式乘法展开
    # 初始化多项式为 [1.0] (对应常数 1)
    den_poly = [1.0 + 0.0im]

    for p in poles
        # 多项式乘法: (coeff...)*s - (coeff...)*p
        # 相当于卷积: conv(den_poly, [1, -p])
        new_poly = zeros(ComplexF64, length(den_poly) + 1)
        # 移位加
        new_poly[1:end-1] += den_poly    # 对应 * s
        new_poly[2:end]   -= den_poly * p # 对应 * -p
        den_poly = new_poly
    end

    # 理论上巴特沃兹多项式系数全是实数，取实部即可
    return real.(den_poly)
end

# ==========================================
#  2. 主程序
# ==========================================

N_order = 4
Wc = 1.0

println("正在设计 4 阶巴特沃兹滤波器 (Wc=1)...")
den_coeffs = design_butterworth_analog(N_order, Wc)

# 打印传递函数
println("\n========================================")
println("       设计结果: 传递函数 H(s)")
println("========================================")
println("H(s) = 1 / D(s)")
println("D(s) 系数 (从高阶 s^4 到 s^0):")
println(round.(den_coeffs, digits=4))
println("\n数学表达式:")
print("H(s) = 1 / ( s^4")
@printf(" + %.4fs^3", den_coeffs[2])
@printf(" + %.4fs^2", den_coeffs[3])
@printf(" + %.4fs",   den_coeffs[4])
@printf(" + %.4f )",  den_coeffs[5])
println("\n")

# ==========================================
#  3. 计算增益响应并绘图
# ==========================================

# 定义频率范围 (模拟频率 rad/s)
w_axis = range(0, 3.0, length=500)

# 直接利用模平方公式计算幅度，这样最精确
# |H(jw)| = 1 / sqrt(1 + (w/Wc)^(2N))
mag_response = [1.0 / sqrt(1 + (w/Wc)^(2*N_order)) for w in w_axis]

# 转换为 dB
```

```
gain_db = 20 * log10.(mag_response)

figure("Question 37: Butterworth Response")

# 1. 绘制增益曲线
plot(w_axis, gain_db, "b-", linewidth=2, label="Gain Response")
hold("on")

# 2. 标记 3-dB 截止点
# 在 w=1 处，增益应为 -3.01 dB
plot([1.0], [-3.0103], "ro", markersize=8, label="Cutoff (1 rad/s, -3dB)")

# 3. 绘制辅助线
plot([0, 1.0], [-3.01, -3.01], "r--", linewidth=1) # 横向虚线
plot([1.0, 1.0], [-50, -3.01], "r--", linewidth=1) # 纵向虚线

title("Gain Response of 4th-Order Butterworth Low-Pass Filter")
xlabel("Frequency (rad/s)")
ylabel("Gain (dB)")
grid("on")
legend("on")

# 设置显示范围
ylim([-40, 5])
xlim([0, 3])

hold("off")
```

38.已知系统的零极点分别如下:

$$z_1 = 2.2, z_2 = -1 + j, z_3 = -1 - j, z_4 = 1.4$$

$$p_1 = 3.7 + j2, p_2 = 3.7 - j2, p_3 = -2.1 - j, p_4 = -2.1 + j$$

求系统的系统函数 H(z)。

```
using TyPlot      # Syslab 原生绘图
using LinearAlgebra
using Printf

# ==========================================
#  1. 辅助函数：从根构建多项式系数
# ==========================================
# 相当于 MATLAB 的 poly() 函数
function poly_from_roots(roots_val)
    # 初始化多项式为 [1] (对应最高次项系数)
    coeffs = [1.0 + 0.0im]

    for r in roots_val
        # 多项式卷积: (coeff...)*z - (coeff...)*r
        # 原系数左移一位 (乘z) + 原系数乘-r
        new_len = length(coeffs) + 1
        new_coeffs = zeros(ComplexF64, new_len)

        new_coeffs[1:end-1] += coeffs      # z 的高次项
        new_coeffs[2:end]   -= coeffs .* r  # 常数项/低次项

        coeffs = new_coeffs
    end

    # 由于共轭根的存在，理论上虚部应抵消为0，取实部即可
    return real.(coeffs)
end

# ==========================================
#  2. 定义零点和极点
# ==========================================
# 零点 Zeros
z_roots = [
    2.2,
    -1.0 + 1.0im,
    -1.0 - 1.0im,
    1.4
]

# 极点 Poles
```

```
p_roots = [
    3.7 + 2.0im,
    3.7 - 2.0im,
    -2.1 - 1.0im,
    -2.1 + 1.0im
]

# ========================================
#  3. 计算系统函数系数
# ========================================
b = poly_from_roots(z_roots) # 分子系数
a = poly_from_roots(p_roots) # 分母系数

println("========================================")
println("         第 38 题计算结果")
println("========================================")

println("系统函数 H(z) 的系数 (标准形式):")
println("H(z) = (b0 + b1*z^-1 + ... ) / (a0 + a1*z^-1 + ...)")
println("----------------------------------------")

println("[分子系数 b]:")
println(round.(b, digits=4))

println("\n[分母系数 a]:")
println(round.(a, digits=4))

println("\n数学表达式 H(z):")
print("H(z) = ( 1")
@printf(" %.4f z^-1 + %.4f z^-2 + %.4f z^-3 + %.4f z^-4 )\n", b[2], b[3], b[4], b[5])
print("       ------------------------------------------------------------------------\n")
print("       ( 1")
@printf(" %.4f z^-1 + %.4f z^-2 + %.4f z^-3 + %.4f z^-4 )\n", a[2], a[3], a[4], a[5])


# ========================================
#  4. 绘制零极点图 (验证用)
# ========================================
figure("Question 38: Pole-Zero Plot")
hold("on")

# 画单位圆
theta = range(0, 2pi, length=200)
plot(cos.(theta), sin.(theta), "k--", linewidth=1.5, label="Unit Circle")

# 画零点
plot(real(z_roots), imag(z_roots), "bo", markersize=9, label="Zeros")
# 画极点
plot(real(p_roots), imag(p_roots), "rx", markersize=10, label="Poles")

title("Pole-Zero Plot")
xlabel("Real Axis")
ylabel("Imaginary Axis")
grid("on")
legend("on")
axis("equal")

# 调整视野以包含所有点
max_val = maximum(abs.([z_roots; p_roots])) * 1.2
xlim([-max_val, max_val])
ylim([-max_val, max_val])

hold("off")
```

39.设计椭圆 IIR 数字低通滤波器，其性能指标为：通带截止频率为 1000Hz，阻带截止频率为 1250Hz，通带波纹为 0.4dB，最小阻带衰减为 45dB，抽样频率为 5000Hz。绘制所设计的滤波器增益响应。

```
using DSP
using TyPlot

# ============================
# 1. 定义设计指标
# ============================
Fs = 5000.0       # 抽样频率 (Hz)
fp = 1000.0       # 通带截止频率 (Hz)
fs_stop = 1250.0  # 阻带截止频率 (Hz)
```

```
Rp = 0.4          # 通带波纹 (dB)
Rs = 45.0         # 最小阻带衰减 (dB)

# ============================
# 2. 滤波器设计
# ============================
nyquist = Fs / 2
wp = fp / nyquist
ws = fs_stop / nyquist

println("正在计算滤波器参数...")
# 计算阶数
N, Wn = ellipord(wp, ws, Rp, Rs, domain=:z)
println("滤波器阶数 N: ", N)

# 设计滤波器
# ！！！ 修改处：显式使用 DSP.Elliptic 以避免命名冲突！！！
filt_design = digitalfilter(Lowpass(Wn), DSP.Elliptic(N, Rp, Rs))

# ============================
# 3. 计算频率响应
# ============================
nb_points = 1024
freq_range = range(0, nyquist, length=nb_points)

# 计算复数响应
H = freqz(filt_design, freq_range, Fs)

# 转换为 dB
mag_db = 20 * log10.(abs.(H))

# ============================
# 4. 使用 TyPlot 绘图
# ============================
println("正在使用 TyPlot 绘图...")

# 创建图形窗口
figure()

# 1. 绘制主响应曲线 (蓝色实线)
plot(freq_range, mag_db, "b", linewidth=1.5)

# 保持图形，以便叠加辅助线
hold("on")

# 2. 绘制通带波纹限制线 (红色虚线)
plot([0, fp], [-Rp, -Rp], "r--", linewidth=1.5)

# 3. 绘制阻带衰减限制线 (绿色虚线)
plot([fs_stop, nyquist], [-Rs, -Rs], "g--", linewidth=1.5)

# 4. 绘制截止频率竖线 (黑色点线，辅助观察)
plot([fp, fp], [-100, 5], "k:")
plot([fs_stop, fs_stop], [-100, 5], "k:")

# 设置图形属性
title("椭圆 IIR 数字低通滤波器增益响应 (TyPlot)")
xlabel("频率 (Hz)")
ylabel("幅度 (dB)")

# 限制 Y 轴范围方便观察
ylim([-100, 5])
xlim([0, nyquist])

# 打开网格
grid("on")

# 添加图例
legend(["滤波器响应", "通带限制 (-0.4dB)", "阻带限制 (-45dB)", "截止频率标记"])

println("绘图完成。")
```

40. 编写总体均值滤波器程序。原始未受干扰的序列为：$s[n]=3[n(0.8)^n]$，加性噪声信号 d[n] 为随机序列，幅度 0.6，受干扰的序列为：$x[n]=s[n]+d[n]$，绘制噪声序列和 60 次检测结果的总体平均的序列。

```
using TyPlot
using Random

# ============================
# 1. 参数设置
# ============================
n = 0:50              # 时间序列
M = 60                # 平均次数
noise_amp = 0.6       # 噪声幅度

# ============================
# 2. 生成原始信号 s[n]
# ============================
s = 3 .* n .* (0.8) .^ n

# ============================
# 3. 总体均值滤波模拟
# ============================
x_accumulated = zeros(length(n))
x_single = zeros(length(n))

println("正在进行计算...")

for i in 1:M
    # 生成噪声: 范围 [-0.6, 0.6]
    d = noise_amp .* (2 .* rand(length(n)) .- 1)

    # 受干扰信号
    x_current = s .+ d

    # 【修正点】记录第1次结果
    if i == 1
        # 使用 [:] 进行原地更新，确保修改的是外部变量
        x_single[:] = x_current
    end

    # 累加信号 (使用 global 也是一种方法，但对于数组推荐用原地更新)
    global x_accumulated += x_current
end

# 计算平均值
x_avg = x_accumulated ./ M

# ============================
# 4. 绘图
# ============================
figure()

# --- 子图1: 单次噪声序列 ---
subplot(2, 1, 1)
plot(n, x_single, "b.-")
title("单次受干扰信号 (带噪声)")
xlabel("n")
ylabel("幅度")
grid("on")
xlim([0, 50])

# --- 子图2: 总体平均序列 ---
subplot(2, 1, 2)
plot(n, x_avg, "r.-", linewidth=1.5)
hold("on")
plot(n, s, "k--", linewidth=1.0) # 原始信号参考
title("60次检测结果的总体平均序列")
xlabel("n")
ylabel("幅度")
legend(["总体平均信号", "原始信号(参考)"])
grid("on")
xlim([0, 50])

println("绘图完成。现在你应该能看到上面的图有明显的噪声毛刺了。")
```