

**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 $0.4\pi$ ，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
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
using Optim
using LinearAlgebra
using Statistics

# =====
# 1. 辅助函数：构造全通滤波器
# =====
# 根据极点半径(r)和角度(theta)构建级联的全通滤波器 (ZeroPoleGain形式)
function make_allpass(params)
    n_sections = div(length(params), 2)
    total_filter = nothing

    for i in 1:n_sections
        r = params[2*i - 1]
        theta = params[2*i]

        # 共轭极点对
        p = r * exp(im * theta)
        poles = [p, conj(p)]
        # 全通滤波器的零点是极点的倒数共轭
        zeros_vec = [1/conj(p), 1/p]

        # 增益设为 r^2 以保持单位增益
        section = ZeroPoleGain(zeros_vec, poles, 1.0)

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

    return total_filter
end

# =====
# 2. 优化函数
# =====
function optimize_group_delay(iir_filter, wp, N_ap)
    println("正在优化全通均衡器参数...")

    n_sections = div(N_ap, 2)
    # 优化频率范围: 0 到 截止频率 (通带)
    w_pass = range(0, stop=wp*pi, length=100)

    # 计算原始 IIR 的群延时
    tau_iir = grpdelay(iir_filter, w_pass)
    mean_tau_iir = mean(tau_iir)

    # 目标函数: 最小化 (总群延时 - 平均值) 的标准差
    function cost_function(params)
        try
            ap_filter = make_allpass(params)
            tau_ap = grpdelay(ap_filter, w_pass)
            tau_total = tau_iir + tau_ap
            # 我们希望群延时尽可能平坦, 即标准差最小
            return std(tau_total)
        catch
            return Inf
        end
    end
```

```

end

# 初始猜测 [r1, theta1, r2, theta2, ...]
initial_params = Float64[]
for i in 1:n_sections
    push!(initial_params, 0.5 + 0.1*i) # r
    push!(initial_params, 0.1*pi * i) # theta
end

# 约束: 半径 0~1 (稳定), 角度 0~pi
lower_bounds = repeat([0.0, 0.0], n_sections)
upper_bounds = repeat([0.99, pi], n_sections)

# 使用 Fminbox + BFGS 优化
res = optimize(cost_function, lower_bounds, upper_bounds, initial_params, Fminbox(BFGS()), Optim.Options(time_limit=15.0))

best_params = Optim.minimizer(res)
println("优化结束。最终代价 (std): ", Optim.minimum(res))
return make_allpass(best_params)
end

# =====
# 3. 绘图函数
# =====
function plot_results(iir_filter, ap_filter, wp, ws)
    # --- 准备数据 ---
    # 频率轴: 全频段 [0, pi] 用于幅频响应, 通带 [0, wp*pi] 用于群延时细节
    n_points = 1000
    w_full = range(0, stop=pi, length=n_points)
    w_pass = range(0, stop=wp*pi, length=n_points)

    # 1. 计算幅频响应 (Magnitude)
    # 使用 freqz 计算复数响应
    H_iir = freqz(iir_filter, w_full)
    H_ap = freqz(ap_filter, w_full)
    H_total = H_iir .* H_ap

    mag_iir = abs.(H_iir)
    mag_total = abs.(H_total)

    # 归一化 (以 DC 增益为基准)
    mag_iir ./= mag_iir[1]
    mag_total ./= mag_total[1] # 级联后可能会有微小增益变化, 重新归一化

    # 2. 计算群延时 (Group Delay) - 仅关注通带和过渡带
    tau_iir = grpdelay(iir_filter, w_pass)
    tau_ap = grpdelay(ap_filter, w_pass)
    tau_total = tau_iir + tau_ap

    # --- 开始绘图 ---
    figure("IIR Filter & Group Delay Equalization")
    clf()

    # 子图 1: 幅频响应 (Magnitude)
    subplot(2, 1, 1)
    hold("on")
    plot(w_full/pi, mag_iir, "b-", linewidth=1.5, label="原始低通 IIR")
    plot(w_full/pi, mag_total, "r-", linewidth=1.5, label="级联后 (IIR + 全通)")

    # 标记通带截止频率
    plot([wp, wp], [0, 1.1], "k:", linewidth=1)

    # 样式设置
    ylabel("归一化幅值")
    title("1. 幅频响应对比 (验证全通特性)")
    grid("on")
    legend()
    xlim([0, 1])
    ylim([0, 1.1])

    # 可以在图上标注一下, 全通不改变幅度
    # 修复: 将 color="gray" 改为十六进制 "#808080", 因为 TyPlot 不支持 "gray" 名称
    text(0.5, 0.5, "注意: 红虚线应与蓝实线重合\n(全通滤波器不改变幅度谱)", fontsize=8, color="#808080")

    # 子图 2: 群延时 (Group Delay)
    subplot(2, 1, 2)
    hold("on")
    plot(w_pass/pi, tau_iir, "b-", linewidth=1.5, label="原始群延时")
    plot(w_pass/pi, tau_total, "r-", linewidth=2.0, label="均衡后总群延时")

    # 绘制平均延时参考线
    avg_delay = mean(tau_total)
    plot([0, wp], [avg_delay, avg_delay], "g-", linewidth=1, label="平均延时 ($(round(avg_delay, digits=1)))")

```

```

# 样式设置
xlabel("归一化频率 ( $\times\pi$  rad/sample)")
ylabel("群延时 (samples)")
title("2. 通带群延时均衡效果")
grid("on")
legend()
xlim([0, wp]) # 聚焦通带

# 自动调整Y轴范围以展示细节
y_min = minimum(tau_total) * 0.8
y_max = maximum(tau_iir) * 1.1
ylim([y_min, y_max])

hold("off")
end

# =====
# 主程序
# =====
println("=== 开始设计 ===")

# 1. 滤波器指标
wp = 0.4 # 通带截止 (x pi)
ws = 0.5 # 阻带起始 (x pi)
Rp = 0.6 # dB
Rs = 32.0 # dB
N_iir = 3 # IIR 阶数

# 2. 设计 IIR 低通滤波器
println("1. 设计 3阶 椭圆低通滤波器...")
# 显式指定 design_method
response = Lowpass(wp)
method = DSPElliptic(N_iir, Rp, Rs)
iir_filter = digitalfilter(response, method)

# 3. 设计全通均衡器
println("2. 设计 6阶 全通均衡器 (优化中)...")
N_ap = 6
ap_filter = optimize_group_delay(iir_filter, wp, N_ap)

# 4. 绘图
println("3. 绘制分析图...")
plot_results(iir_filter, ap_filter, wp, ws)

println("=== 完成 ===")

```

**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。

```
using LinearAlgebra
using Printf

println("=== Q3: 部分分式展开 (对应 MATLAB residuez) ===")

# -----
# 1) 题目系数 (按 z^-1 升幂: b0 + b1 z^-1 + ...)
# -----
b = [1.0, -0.2, 0.5]      # 1 - 0.2 z^-1 + 0.5 z^-2
a = [1.0, 3.2, 1.5, -0.8, 1.4] # 1 + 3.2 z^-1 + 1.5 z^-2 - 0.8 z^-3 + 1.4 z^-4

# -----
# 2) 工具: 降幂多项式求根 c0 z^n + c1 z^(n-1)+...+cn
# -----
function roots_desc(c::AbstractVector{<:Real})
    n = length(c) - 1
    @assert n >= 1 "多项式阶数必须≥1"
    c0 = c[1]
    @assert abs(c0) > 1e-12 "最高次系数不能为0"

    cnorm = c ./ c0 # 归一化最高次为1

    C = zeros(ComplexF64, n, n)
    for i in 2:n
        C[i, i-1] = 1.0
    end
    # companion 最后一列 = -[cn, c(n-1), ..., c1]^T
    C[:, n] .= -reverse(cnorm[2:end])
    eigvals(C)
end

# 多项式求值: 降幂系数
polyval_desc(c::AbstractVector, z) = begin
    y = 0.0 + 0.0im
    for i in eachindex(c)
        y = y*z + c[i]
    end
    y
end

# 降幂导数系数
function polyder_desc(c::AbstractVector{<:Real})
    n = length(c) - 1
    d = zeros(Float64, n)
    for i in 1:n
        d[i] = c[i] * (n - (i - 1))
    end
    d
end

# -----
# 3) 把 A(z^-1) 转成 z 平面 D(z)=z^M A(z^-1) 并求极点
# A(z^-1)=1+a1 z^-1+...+aM z^-M
# => D(z)=z^M + a1 z^(M-1) + ... + aM
```

```

# 其降幂系数刚好是 a 本身
# -----
M = length(a) - 1
D_desc = a          # z^4 + 3.2 z^3 + 1.5 z^2 - 0.8 z + 1.4
p = roots_desc(D_desc)  # z 平面极点

# -----
# 4) 留数计算 (对应  $H(z) = \sum r_i / (1 - p_i z^{-1})$ )
#
# 令  $N(z) = z^M B(z^{-1})$ 
#  $B(z^{-1}) = b_0 + b_1 z^{-1} + \dots + b_K z^{-K}$ 
#  $\Rightarrow N(z) = b_0 z^M + b_1 z^{(M-1)} + \dots + b_K z^{(M-K)} + \dots$  (不足补0)
#
# 对简单极点, 有公式:
#  $r_i = N(p_i) / (p_i * D'(p_i))$ 
# -----
# 构造 N(z) 的降幂系数 (长度 M+1)
N_desc = zeros(Float64, M+1)
for k in 0:(length(b)-1)
    N_desc[1+k] = b[1+k] # b0,b1,b2...
end
# N_desc 现在是 [b0,b1,b2,0,0]  $\Rightarrow z^4 - 0.2 z^3 + 0.5 z^2$ 

Dder_desc = polyder_desc(D_desc)

r = ComplexF64[]
for pi in p
    ri = polyval_desc(N_desc, pi) / (pi * polyval_desc(Dder_desc, pi))
    push!(r, ri)
end

# -----
# 5) 打印: 复数留数形式 (与 residues 同型)
# -----
println("\n--- 极点 p (z 平面) ---")
for (i, val) in enumerate(p)
    @printf("p%d = %.6f %+.6fim\n", i, real(val), imag(val))
end

println("\n--- 留数 r (使  $H(z) = \sum r/(1-p z^{-1})$ ) ---")
for (i, val) in enumerate(r)
    @printf("r%d = %.6f %+.6fim\n", i, real(val), imag(val))
end

println("\n--- 部分分式展开 (复数形式) ---")
println("H(z) = ")
for i in eachindex(r)
    @printf(" %.6f %+.6fim / (1 - (%.6f %+.6fim) z^-1)\n",
        real(r[i]), imag(r[i]), real(p[i]), imag(p[i]))
end

# -----
# 6) 可选: 把共轭对合并成“实系数二阶项” (更像手写答案)
# 若 (r,p) 与 (conj(r),conj(p)):
#  $r/(1-pq) + r^*/(1-p^*q) =$ 
#  $(2\text{Re}(r) - 2\text{Re}(r^*\text{conj}(p)) q) / (1 - 2\text{Re}(p) q + |p|^2 q^2), q = z^{-1}$ 
# -----
function pair_conj_indices(vals::Vector{ComplexF64}; tol=1e-6)
    used = falses(length(vals))
    pairs = Tuple{Int,Int}[]
    singles = Int[]
    for i in eachindex(vals)
        used[i] && continue
        if abs(imag(vals[i])) < tol
            push!(singles, i); used[i]=true
        else
            j = findfirst(j -> !used[j] && j!=i && abs(vals[j]-conj(vals[i])) < 1e-4, eachindex(vals))
            if j == nothing
                push!(singles, i); used[i]=true
            else
                push!(pairs, (i,j)); used[i]=true; used[j]=true
            end
        end
    end
    return pairs, singles
end

pairs, singles = pair_conj_indices(p)

println("\n--- 合并共轭后的实系数形式 ---")
for (i,j) in pairs
    ri, pi = r[i], p[i]

```

```

b0 = 2*real(ri)
b1 = -2*real(ri*conj(pi))
a1 = -2*real(pi)
a2 = abs(pi)^2
@printf(" (%.6f %+ .6f z^-1) / (1 %+ .6f z^-1 %+ .6f z^-2)\n", b0, b1, a1, a2)
end
for i in singles
    # 实极点对应一阶项
    @printf(" (%.6f) / (1 - (%.6f) z^-1)\n", real(ri[i]), real(pi[i]))
end

# -----
# 7) 数值校验: 在单位圆上采样对比原系统与展开式
# (包进函数, 避免 Mworks soft scope 报错)
# -----
function validate_pf(b, a, r, p)
    function H_from_ba(b,a,ω)
        q = exp(-1im*ω) # z^-1
        num = sum(b[k]*q^(k-1) for k in 1:length(b))
        den = sum(a[k]*q^(k-1) for k in 1:length(a))
        num/den
    end

    function H_from_rp(r,p,ω)
        q = exp(-1im*ω)
        s = 0.0 + 0.0im
        for i in eachindex(r)
            s += r[i] / (1 - p[i]*q)
        end
        s
    end

    ws = range(0, 2π, length=2048)
    err = 0.0
    for ω in ws
        err = max(err, abs(H_from_ba(b,a,ω) - H_from_rp(r,p,ω)))
    end
    @printf("\n校验: max |H_orig - H_pf| = %.3e\n", err)
end

validate_pf(b, a, r, p)

```

**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。。**

```

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_dB = 0.5      # 通带波纹 (dB)
Rs_dB = 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)

# 将 dB 指标转换为线性幅度指标以便后续绘图和验证
# 通带最小幅度: 10^(-0.5/20)
mag_pass_min = 10^(-Rp_dB / 20)
# 阻带最大幅度: 10^(-43/20)
mag_stop_max = 10^(-Rs_dB / 20)

println("指标参数:")
println(" 通带波纹: $Rp_dB dB (线性幅度 > $(round(mag_pass_min, digits=4)))")
println(" 阻带衰减: $Rs_dB dB (线性幅度 < $(round(mag_stop_max, digits=5)))")
println(" 通带频率: $(wp)π")
println(" 阻带频率: $(ws)π")

# =====
# 2. 计算最小阶数 N
# =====

```

```

# 使用双线性变换的模拟原型法计算阶数
# 1. 预畸变: 将数字频率映射到模拟频率 Omega = tan(w/2)
Omega_p = tan(wp * pi / 2)
Omega_s = tan(ws * pi / 2)

# 2. 计算高通滤波器的选择性因子 (映射到低通原型)
# 对于高通 HP -> 低通 LP 变换:  $\lambda = \Omega_p / \Omega$ 
# 阻带边缘对应的低通原型频率  $\lambda_s = \Omega_p / \Omega_s$ 
lambda_s = Omega_p / Omega_s

# 3. 切比雪夫阶数公式
numerator = acosh(sqrt((10^(0.1 * Rs_dB) - 1) / (10^(0.1 * Rp_dB) - 1)))
denominator = acosh(lambda_s)
N = ceil( (int, numerator / denominator)

println("\n计算结果:")
println(" 模拟频率比  $\lambda_s$ : $(round(lambda_s, digits=4))")
println(" 所需最小阶数 N: $N")

# =====
# 3. 设计滤波器
# =====
# Highpass(Wn) 中的 Wn 是归一化截止频率
# Chebyshev1(N, ripple) 中的 ripple 是 dB
response_type = Highpass(wp)
design_method = Chebyshev1(N, Rp_dB)

filter_obj = digitalfilter(response_type, design_method)
println(" 滤波器设计完成。")

# =====
# 4. 计算频率响应并绘图 (线性幅度)
# =====
# 定义频率向量
w_range = range(0, stop=pi, length=1024)
h_resp = freqz(filter_obj, w_range)

# 【优化】计算线性幅度 (Linear Magnitude)
mag = abs(h_resp)
# 【优化】转换为数组确保 TyPlot 兼容性
frequencies_normalized = collect(w_range ./ pi)

println("正在绘制线性增益响应...")

figure("Chebyshev Type I Highpass Filter", figsize=(10, 8))

# 1. 绘制幅频响应曲线
plot(frequencies_normalized, mag, "b", linewidth=2, label="幅频响应 |H(e^jw)|")
hold("on")

# 2. 绘制指标限制线 (线性坐标)

# (A) 通带区域 (0.75 - 1.0)
# 上限: 1.0 (0dB)
plot([wp, 1.0], [1.0, 1.0], "g--", label="通带上限 (1.0)")
# 下限:  $10^{(-0.5/20)} \approx 0.944$ 
plot([wp, 1.0], [mag_pass_min, mag_pass_min], "g--", label="通带下限 (-0.5dB)")

# (B) 阻带区域 (0 - 0.35)
# 上限:  $10^{(-43/20)} \approx 0.007$ 
plot([0, ws], [mag_stop_max, mag_stop_max], "r--", label="阻带上限 (-43dB)")

# (C) 截止频率垂直标记
plot([wp, wp], [0, 1.1], "k:", label="通带截止 (0.75 $\pi$ )")
plot([ws, ws], [0, 1.1], "k:", label="阻带截止 (0.35 $\pi$ )")

# 设置图形属性
title("切比雪夫 I 型高通滤波器 (N=$N) - 线性幅度响应")
xlabel("归一化频率 ( $\times \pi$  rad/sample)")
ylabel("幅度 (归一化 0-1)")
xlim(0, 1)
ylim(0, 1.1) # 线性坐标下, 稍微多出一小空间看清楚 1.0

grid(true)
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

# 自动检查并加载 TyPlot 绘图库
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 (MWorks/Julia) ===")
```

```

# =====
# 1. 辅助函数: 处理用户输入
# =====
function get_user_input(prompt::String, parse_type::Type, validator::Function=(x->true))
    while true
        print(prompt)
        flush(stdout) # 确保提示符立即显示
        try
            str = strip(readline())
            if isempty(str) continue end
            val = parse(parse_type, str)
            if validator(val)
                return val
            else
                println("输入数值超出合理范围, 请重新输入。")
            end
        catch
            println("输入格式无效, 请输入一个有效的 $(parse_type)。")
        end
    end
end

# =====
# 2. 核心算法: 设计滤波器
# =====
...

design_chebyshev_lowpass(N, fc_3db, Rp_db)

参数:
- N: 阶数
- fc_3db: 用户输入的 3dB 截止频率
- Rp_db: 通带波纹 (dB)
返回:
- poles: S平面极点
- numerator: 分子系数 (增益)
- fp_edge: 计算出的通带边缘频率 (在此频率处衰减量为 Rp)
...

function design_chebyshev_lowpass(N::Int, fc_3db::Float64, Rp_db::Float64)
    # 1. 计算波纹因子 epsilon
    epsilon = sqrt(10^(Rp_db / 10.0) - 1.0)

    # 2. 计算缩放因子
    # 切比雪夫滤波器的 3dB 频率与通带边缘频率的关系:
    #  $\omega_c = \omega_p * \cosh(1/N * \operatorname{acosh}(1/\epsilon))$ 
    scaling_factor = cosh((1.0 / N) * acosh(1.0 / epsilon))

    # 用户输入的 fc_3db 就是 -3dB 频率
    wp_3db = 2 * pi * fc_3db # 3dB 频率的角频率
    wp_edge = wp_3db / scaling_factor # 通带边缘的角频率
    fp_edge = wp_edge / (2 * pi) # 通带边缘频率 (Hz)

    # 3. 计算 S 平面极点 (分布在椭圆上)
    mu = asinh(1.0 / epsilon) / N
    poles = ComplexF64[]

    for k in 1:N
        # 切比雪夫极点角度公式
        theta = (2*k - 1) * pi / (2*N)

        sigma = -sinh(mu) * sin(theta)
        omega = cosh(mu) * cos(theta)

        # 使用通带边缘频率进行缩放
        s_k = wp_edge * (sigma + im * omega)
        push!(poles, s_k)
    end

    # 4. 计算归一化增益 (分子系数)
    denom_at_0 = real(prod(-poles))

    if N % 2 == 1 # 奇数阶
        numerator = denom_at_0 # 直流增益 = 1
    else # 偶数阶
        numerator = denom_at_0 / sqrt(1 + epsilon^2)
    end

    return poles, numerator, fp_edge
end

# 计算复数频率响应
function calc_complex_response(poles, numerator, f_array)

```

```

w_array = 2 * pi .* f_array
s_array = im .* w_array
# 传递函数 H(s) = K / product(s - pk)
H = map(s -> numerator / prod(s.- poles), s_array)
return H
end

# =====
# 3. 主程序
# =====

function main()
    try
        # --- 获取参数 ---
        N = get_user_input("请输入滤波器阶数 N (整数, >0): ", Int, x -> x > 0)
        fc_input = get_user_input("请输入 3-dB 截止频率 fc (Hz, >0): ", Float64, x -> x > 0)
        Rp = get_user_input("请输入通带波纹 Rp (dB, >0): ", Float64, x -> x > 0)

        println("\n-----")
        println("正在计算滤波器参数...")
        println(" 阶数 (N)      : $N")
        println(" 3dB 截止频率   : $fc_input Hz")
        println(" 通带波纹       : $Rp dB")
        println("-----")

        # --- 设计滤波器 ---
        poles, numerator, fp_calc = design_chebyshev_lowpass(N, fc_input, Rp)

        println("计算完成:")
        println(" 通带边缘频率 (fp): $(round(fp_calc, digits=2)) Hz (波纹终止点)")

        # --- 准备绘图数据 ---
        # 1. 全景数据 (0 到 3倍截止频率)
        f_full = range(0, stop=3*fc_input, length=1000)
        H_full = calc_complex_response(poles, numerator, f_full)
        mag_full_db = 20 .* log10.(abs.(H_full))

        # 2. 细节数据 (0 到 1.2倍通带边缘, 用于观察波纹)
        f_zoom = range(0, stop=1.2*fp_calc, length=1000)
        H_zoom = calc_complex_response(poles, numerator, f_zoom)
        mag_zoom_db = 20 .* log10.(abs.(H_zoom))

        # --- 绘图 ---
        println("\n正在绘制图形...")

        # 子图 1: 全景响应
        subplot(2, 1, 1)
        plot(f_full, mag_full_db, "b-", linewidth=2, label="增益响应")
        hold("on")
        # 标记 3dB 点
        plot([fc_input, fc_input], [-100, 10], "r--", linewidth=1.5, label="3dB 截止频率")
        plot([0, maximum(f_full)], [-Rp, -Rp], "g:", linewidth=1.5, label="波纹界限 (-$(Rp)dB)")

        title("切比雪夫 I 型滤波器: 整体幅频响应")
        ylabel("增益 (dB)")
        grid("on")
        legend()

        # 计算最小增益值, 但确保不会低于 -60dB
        min_gain = minimum(mag_full_db)
        y_lower = max(-60.0, min_gain)
        y_upper = Rp + 2
        ylim(y_lower, y_upper) # 限制Y轴范围以便观察衰减

        # 子图 2: 通带细节放大
        subplot(2, 1, 2)
        plot(f_zoom, mag_zoom_db, "b-", linewidth=2, label="通带波纹")
        hold("on")
        # 绘制 0dB 和 -Rp dB 参考线
        plot([0, maximum(f_zoom)], [0, 0], "k--", linewidth=1)
        plot([0, maximum(f_zoom)], [-Rp, -Rp], "g--", linewidth=1, label="波纹下限")

        title("细节放大: 通带内的波纹特性")
        xlabel("频率 (Hz)")
        ylabel("增益 (dB)")
        grid("on")

        # 根据滤波器阶数设置合适的Y轴范围
        if N % 2 == 1
            # 奇数阶: DC处增益为0dB
            ylim(-Rp*1.2, 0.5)
        else
            # 偶数阶: DC处增益为-Rp dB

```

```

        ylim(-Rp*1.5, 0.5)
    end

    println("绘图完成。")
    println("请查看绘图窗口：")
    println(" - 上图展示了整体的滤波衰减效果。")
    println(" - 下图放大了通带部分，您可以清晰地看到波纹抖动。")

    catch e
        println("\n发生错误: $e")
        println("错误类型: $(typeof(e))")
        println("请检查参数设置是否正确。")
    end
end

# 运行主程序
main()

```

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_rational)

println("\n----- 分子与分母 -----")
println("分子 N(z): ", 5.76*num)
println("分母 D(z): ", 5.76*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。**

```
using DSP
using TyPlot
using Printf

println("=== 巴特沃兹 IIR 带通滤波器设计 ===")

# =====
# 1. 定义性能指标
# =====
# 归一化频率 (1.0 =  $\pi$ )
wp = (0.4, 0.6) # 通带范围
ws = (0.3, 0.7) # 阻带范围
Rp_dB = 0.6 # 通带波纹 (dB)
Rs_dB = 35.0 # 最小阻带衰减 (dB)

# 将 dB 指标转换为线性幅度以便绘图
# 通带最小幅度:  $10^{(-0.6/20)}$ 
mag_pass_min = 10^(-Rp_dB / 20)
# 阻带最大幅度:  $10^{(-35/20)}$ 
mag_stop_max = 10^(-Rs_dB / 20)

println("指标参数:")
println(" 通带频率:  $(wp[1]\pi$  到  $(wp[2])\pi$ ")
println(" 阻带频率:  $< (ws[1])\pi$  和  $> (ws[2])\pi$ ")
println(" 通带波纹:  $Rp\_dB$  dB (线性幅度  $> (round(mag\_pass\_min, digits=4))$ )")
println(" 阻带衰减:  $Rs\_dB$  dB (线性幅度  $< (round(mag\_stop\_max, digits=5))$ )")

# =====
# 2. 计算阶数和截止频率
# =====
# buttord 返回最小阶数 N 和 3dB 截止频率 Wn
N, Wn = buttord(wp, ws, Rp_dB, Rs_dB)

println("\n设计结果:")
println(" 滤波器阶数 N:  $N$ ")
println(" 3dB 截止频率 Wn:  $(round(Wn, digits=4))$ ")

# =====
# 3. 设计滤波器
# =====
responsetype = Bandpass(Wn[1], Wn[2])
designmethod = Butterworth(N)
filter_system = digitalfilter(responsetype, designmethod)

println(" 滤波器设计完成。")

# =====
# 4. 计算频率响应并绘图 (线性幅度)
# =====
# 定义频率向量
w_range = range(0, stop=pi, length=1024)
h_resp = freqz(filter_system, w_range)

# 【优化】计算线性幅度
mag = abs(h_resp)
# 【优化】转换为数组确保 TyPlot 兼容性
frequencies_normalized = collect(w_range ./ pi)

println("正在绘制线性增益响应...")

figure("Butterworth IIR Bandpass Filter", figsize=(10, 8))

# 1. 绘制幅频响应曲线
plot(frequencies_normalized, mag, "b", linewidth=2, label="幅频响应  $|H(e^{jw})|$ ")
hold("on")

# 2. 绘制指标限制线 (线性坐标)

# (A) 通带区域 (0.4 - 0.6)
# 上限: 1.0
plot([wp[1], wp[2]], [1.0, 1.0], "g--", label="通带上限 (1.0)")
```

```

# 下限: 10^(-0.6/20)
plot([wp[1], wp[2]], [mag_pass_min, mag_pass_min], "g--", label="通带下限 (-0.6dB)")

# (B) 阻带区域 1 (0 - 0.3)
plot([0, ws[1]], [mag_stop_max, mag_stop_max], "r--", label="阻带上限 (-35dB)")

# (C) 阻带区域 2 (0.7 - 1.0)
plot([ws[2], 1.0], [mag_stop_max, mag_stop_max], "r--")

# (D) 垂直标记线
plot([wp[1], wp[1]], [0, 1.1], "k:", label="通带边界")
plot([wp[2], wp[2]], [0, 1.1], "k:")
plot([ws[1], ws[1]], [0, 1.1], "k--", alpha=0.5, label="阻带边界")
plot([ws[2], ws[2]], [0, 1.1], "k--", alpha=0.5)

# 设置图形属性
title("巴特沃兹 IIR 带通滤波器 (N=$N) - 线性幅度响应")
xlabel("归一化频率 (×π rad/sample)")
ylabel("幅度 (归一化 0-1)")
xlim(0, 1)
ylim(0, 1.1)

grid(true)
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("\nn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  $0.35\pi$ , 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。**

```
import Pkg
# 自动检测并安装缺失的包
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
using Optim
using LinearAlgebra
using Statistics

# =====
# 1. 辅助函数: 构造全通滤波器
# =====
# 根据极点半径(r)和角度(theta)构建级联的全通滤波器 (ZeroPoleGain形式)
function make_allpass(params)
    n_sections = div(length(params), 2)
    total_filter = nothing

    for i in 1:n_sections
        r = params[2*i - 1]
        theta = params[2*i]

        # 共轭极点
        p = r * exp(im * theta)
        poles = [p, conj(p)]
        # 全通滤波器的零点是极点的倒数共轭
        zeros_vec = [1/conj(p), 1/p]

        # 增益设为 r^2 以保持单位增益 (简单起见, 后续绘图归一化处理)
        section = ZeroPoleGain(zeros_vec, poles, 1.0)

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

    return total_filter
end

# =====
# 2. 优化函数
# =====
function optimize_group_delay(iir_filter, wp, N_ap)
    println("正在优化 10 阶全通均衡器参数 (这可能需要一点时间)...")

    n_sections = div(N_ap, 2)
    # 优化频率范围: 0 到 截止频率 (通带)
    w_pass = range(0, stop=wp*pi, length=150) # 增加点数提高精度

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

    # 目标函数: 最小化 (总群延时) 的标准差
    function cost_function(params)
        try
            ap_filter = make_allpass(params)
            tau_ap = grpdelay(ap_filter, w_pass)
            tau_total = tau_iir + tau_ap
            # 我们希望群延时尽可能平坦
            return std(tau_total)
        catch
            return Inf
        end
    end
```

```

end

# 初始猜测 [r1, theta1, r2, theta2, ...]
# 针对 10 阶 (5个节), 我们需要更细致的初始分布
initial_params = Float64[]
for i in 1:n_sections
    # 半径分布在 0.5 ~ 0.9 之间
    push!(initial_params, 0.5 + 0.08*i)
    # 角度均匀分布在通带内, 略微偏移
    push!(initial_params, (i / (n_sections+1)) * wp * pi)
end

# 约束: 半径 0~0.99 (稳定), 角度 0~pi
lower_bounds = repeat([0.0, 0.0], n_sections)
upper_bounds = repeat([0.99, pi], n_sections)

# 使用 Fminbox + BFGS 优化
res = optimize(cost_function, lower_bounds, upper_bounds, initial_params, Fminbox(BFGS()), Optim.Options(time_limit=30.0, iterations=1000))

best_params = Optim.minimizer(res)
println("优化结束. 最终标准差 (std): ", round(Optim.minimum(res), digits=4))
return make_allpass(best_params)
end

# =====
# 3. 绘图函数
# =====

function plot_results(iir_filter, ap_filter, wp, ws)
    # --- 准备数据 ---
    n_points = 1000
    w_full = range(0, stop=pi, length=n_points)
    w_pass = range(0, stop=wp*pi, length=n_points)

    # 1. 计算幅频响应 (Magnitude)
    H_iir = freqz(iir_filter, w_full)
    H_ap = freqz(ap_filter, w_full)
    H_total = H_iir .* H_ap

    mag_iir = abs.(H_iir)
    mag_ap = abs.(H_ap)
    mag_total = abs.(H_total)

    # 归一化 (以 DC 增益为基准)
    mag_iir ./= mag_iir[1]
    # 全通滤波器理论上幅度为1, 但也归一化一下以防万一
    mag_ap ./= mag_ap[1]
    mag_total ./= mag_total[1]

    # 2. 计算群延时 (Group Delay)
    tau_iir = grpdelay(iir_filter, w_pass)
    tau_ap = grpdelay(ap_filter, w_pass)
    tau_total = tau_iir + tau_ap

    # --- 开始绘图 ---
    figure("Q12: IIR Filter & Group Delay Equalization")
    clf()

    # 子图 1: 幅频响应
    subplot(2, 1, 1)
    hold("on")
    plot(w_full/pi, mag_iir, "b-", linewidth=1.5, label="原始低通 IIR")
    # 绘制全通滤波器的幅度 (应该是平直的)
    plot(w_full/pi, mag_ap, "g-", linewidth=1.0, label="全通滤波器 (幅度)")
    plot(w_full/pi, mag_total, "r--", linewidth=1.5, label="级联后总响应")

    # 标记通带截止
    plot([wp, wp], [0, 1.2], "k:", linewidth=1)

    ylabel("归一化幅值")
    title("1. 幅频响应 (验证全通特性)")
    grid("on")
    legend("loc", "best")
    xlim([0, 1])
    ylim([0, 1.2])

    # 子图 2: 群延时
    subplot(2, 1, 2)
    hold("on")
    plot(w_pass/pi, tau_iir, "b-", linewidth=1.5, label="原始群延时")
    plot(w_pass/pi, tau_total, "r-", linewidth=2.0, label="均衡后总群延时")

    # 平均延时参考线

```

```

avg_delay = mean(tau_total)
plot([0, wp], [avg_delay, avg_delay], "g-", linewidth=1, label="平均延时 (${round(avg_delay, digits=1)})")

xlabel("归一化频率 ( $\times\pi$  rad/sample)")
ylabel("群延时 (samples)")
title("2. 通带群延时均衡效果 (10阶全通)")
grid("on")
legend("loc", "best")
xlim([0, wp])

# 自动缩放 Y 轴, 留出一点余量
y_min = minimum(tau_total) * 0.8
y_max = maximum(tau_iir) * 1.1
ylim([y_min, y_max])

hold("off")
end

# =====
# 主程序
# =====
println("=== 第12题: 设计开始 ===")

# 1. 滤波器指标
wp = 0.35 # 通带截止 0.35pi
ws = 0.5 # 阻带起始 (假设值, 题目只给了最小阻带衰减, 通常配合椭圆设计自适应, 或者我们可以给一个合理的过渡带)
# 注意: DSP.Elliptic 不需要显式 ws, 它根据 N, Rp, Rs 设计
Rp = 0.8 # dB
Rs = 35.0 # dB
N_iir = 5 # IIR 阶数

# 2. 设计 IIR 低通滤波器
println("1. 设计 5阶 椭圆低通滤波器...")
response = Lowpass(wp)
method = DSPElliptic(N_iir, Rp, Rs)
iir_filter = digitalfilter(response, method)

# 3. 设计全通均衡器
println("2. 设计 10阶 全通均衡器...")
N_ap = 10
ap_filter = optimize_group_delay(iir_filter, wp, N_ap)

# 4. 绘图
println("3. 绘制分析图...")
plot_results(iir_filter, ap_filter, wp, ws)

println("=== 完成 ===")

```

**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 DSPfilt 进行滤波
y = DSPfilt(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  $0 \leq n < 10$ 
    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 UndefinedVarError: `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
using DSP # 需要安装: 用于 remez 滤波器设计
using FFTW # 需要安装: 用于计算频率响应 (fft)

# =====
# 核心计算函数 1: Hermann 公式估计
# =====
function hermann_estimation(fp, fs, dp, ds, Fs)
    # 1. 归一化频率
    Delta_F = (fs - fp) / Fs

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

    # 3. 计算 D_infinity 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

    # 4. 计算过渡带修正项
    f_correction = 11.01217 + 0.51244 * (log_dp - log_ds)

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

    # 向上取整
    N = ceil(Int, N_exact)
```

```

# 阶数 M
M = N - 1

return N, M, N_exact
end

# =====
# 核心计算函数 2: 设计滤波器并计算响应 (修正版)
# =====
function design_and_analyze_filter(N, fp, fs, Fs)

# 1. 使用 Parks-McClellan (Remez) 算法设计滤波器
# 注意: remez 接受的参数是 阶数(Order) = N-1
M = N - 1

# 定义频带
# ERROR FIX: bands 必须是一个扁平的向量 [start1, end1, start2, end2]
# 对应 desired [val1, val2]
bands = [0.0, fp, fs, Fs/2]
desired = [1.0, 0.0]

# 权重: 为了更好地满足阻带波纹要求, 通常需要给予阻带更高的权重
# weight = [1.0/dp, 1.0/ds] 或 [1.0, dp/ds]
# 这里我们根据题目给定的 dp=0.015, ds=0.021 计算权重比
dp=0.015
ds=0.021
w_pass = 1.0 / dp
w_stop = 1.0 / ds
weights = [w_pass, w_stop]

try
# 调用 remez
h = remez(M, bands, desired, weight=weights, Hz=Fs)

# 2. 计算频率响应 (使用 FFT)
n_fft = 8192 # 增加 FFT 点数使曲线更平滑
H = fft([h; zeros(n_fft - length(h))])

# 3. 生成对应的频率轴
freq_axis = range(0, Fs, length=n_fft)

# 4. 截取前半 (0 ~ Nyquist)
valid_idx = 1:div(n_fft, 2)
f_plot = freq_axis[valid_idx]
mag_plot = abs.(H[valid_idx]) # 幅度

return h, f_plot, mag_plot
catch e
println("设计滤波器失败: $e")
println("请检查是否已安装 DSP 包 (import Pkg; Pkg.add(\"DSP\"))")
return [], [], []
end
end

# =====
# 绘图函数: 绘制规格 + 实际响应
# =====
function plot_filter_results(fp, fs, dp, ds, Fs, N, M, f_response, mag_response)
fmax = Fs / 2

figure("FIR Filter Design Result")
clf()

hold("on")

# --- 1. 绘制公差框 (Tolerance Boxes) ---
# 通带框 (蓝色) - 允许范围 [1-dp, 1+dp]
plot([0, fp], [1+dp, 1+dp], "b-", linewidth=1)
plot([0, fp], [1-dp, 1-dp], "b-", linewidth=1)
plot([fp, fp], [1-dp, 1+dp], "b-", linewidth=1)

# 阻带框 (洋红色) - 允许范围 [0, ds]
plot([fs, fmax], [ds, ds], "m-", linewidth=1)
plot([fs, fs], [0, ds], "m-", linewidth=1)

# --- 2. 绘制实际滤波器响应 ---
if !isempty(mag_response)
plot(f_response, mag_response, "k-", linewidth=1.2, label="实际滤波器响应 (N=$N)")
end

# --- 3. 辅助线 ---

```

```

plot([0, fmax], [1, 1], "k:", linewidth=0.5) # 1.0 参考线
plot([0, fmax], [0, 0], "k-", linewidth=0.5) # 0.0 参考线

# 截止频率竖线
plot([fp, fp], [-0.05, 1+dp+0.05], "k--", alpha=0.3)
plot([fs, fs], [-0.05, ds+0.05], "k--", alpha=0.3)

# --- 4. 标注与设置 ---
xlim([0, fmax])
# 稍微放大Y轴范围以便观察波纹
ylim([-0.05, 1.1])

xlabel("频率 (Hz)")
ylabel("幅度 |H(f)|")
title("FIR 低通滤波器设计结果 (Hermann 估计阶数 N=$N)")
grid("on")
legend("loc", "best")

# 关键点文字
text(fp, -0.08, "fp", ha="center")
text(fs, -0.08, "fs", ha="center")

info_text = "规格:\n通带波纹: $dp\n阻带波纹: $ds"
text(fmax*0.75, 0.6, info_text, bbox=Dict("facecolor"=>"white", "alpha"=>0.8))

hold("off")
end

# =====
# 主程序
# =====

# 1. 定义指标
Fs_val = 5000.0
fp_val = 1500.0
fs_val = 1800.0
dp_val = 0.015
ds_val = 0.021

println("=====")
println("    Hermann FIR 滤波器设计")
println("=====")

# 2. 计算阶数
N_est, M_est, N_ex = hermann_estimation(fp_val, fs_val, dp_val, ds_val, Fs_val)

println("Hermann 估算结果:")
println(" 精确长度 N_exact = $(round(N_ex, digits=4))")
println(" 取整长度 N      = $N_est")
println(" 滤波器阶数 M      = $M_est")
println("-----")

# 3. 设计滤波器并获取响应数据
h_coef, f_data, mag_data = design_and_analyze_filter(N_est, fp_val, fs_val, Fs_val)

if isempty(h_coef)
    println("警告: 未能生成滤波器数据, 仅绘制规格框。")
else
    println("滤波器设计成功。正在绘制响应图...")
end

# 4. 绘图
plot_filter_results(fp_val, fs_val, dp_val, ds_val, Fs_val, N_est, M_est, f_data, mag_data)
println("=====")

```

**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
using Random

# =====
# 1) 5 点中值滤波器 (边界: 缩窗)

```

```

# =====
...

median_filter(signal, L)

一维中值滤波器（窗口长度 L 必须为奇数）。
边界处采用“缩窗”（不补零、不镜像），避免边缘失真。
...

function median_filter(signal::AbstractVector{T}, L::Int) where {T<:Real}
    @assert L > 0 && isodd(L) "窗口长度 L 必须为正奇数（例如 5）"

    N = length(signal)
    y = Vector{Float64}(undef, N)
    half = L ÷ 2

    @inbounds for i in 1:N
        a = max(1, i - half)
        b = min(N, i + half)
        @views y[i] = median(signal[a:b])
    end
    return y
end

# =====
# 2) 题目数据：s[n] 与噪声 d[n]
# =====
N = 40
n = collect(0:N-1)          # 用 collect 避免绘图函数对 Range 支持不一致

# s[n] = 3 * n * (0.8)^n
s = 3.0 .* n .* (0.8).^ n

# d[n]：幅度 0.6 的随机序列（均匀分布在 [-0.6, 0.6]）
noise_amp = 0.6
Random.seed!(2026)          # 固定随机种子：每次运行结果一致，便于验收/调参
d = noise_amp .* (2 .* rand(N) .- 1)

# 受干扰序列 x[n]
x = s .+ d

# =====
# 3) 5 点中值滤波
# =====
L = 5
y = median_filter(x, L)

# =====
# 4) 绘图：按题意分别画 40 点“受干扰序列”和“滤波输出”
# （并额外用虚线叠加 s[n] 方便对比，噪声会更明显）
# =====
figure("Median Filter (L=5)", figsize=(10, 8))

# (1) 受干扰序列
subplot(4, 1, 1)
# TyPlot 的 baselfmt 不支持 " "（空格格式）。这里先画 stem，然后把 baseline 隐藏掉。
h1 = stem(n, x, markerfmt="bo", linefmt="b-", label="x[n]=s[n]+d[n]")
try
    c1 = (h1 isa AbstractVector && length(h1) == 1) ? h1[1] : h1
    if PyCall.pyhasattr(c1, "baseline")
        c1.baseline.set_visible(false)
    end
catch
end
plot(n, s, "g--", linewidth=2, alpha=0.8, label="s[n] (无噪声)")

title("输入信号")
ylabel("幅度")
legend(loc="best")
grid(true)

# (1) 受干扰序列
subplot(4, 1, 2)
# TyPlot 的 baselfmt 不支持 " "（空格格式）。这里先画 stem，然后把 baseline 隐藏掉。
h2 = stem(n, x, markerfmt="bo", linefmt="b-", label="x[n]=s[n]+d[n]")
try
    c2 = (h2 isa AbstractVector && length(h2) == 1) ? h2[1] : h2
    if PyCall.pyhasattr(c2, "baseline")
        c2.baseline.set_visible(false)
    end
catch
end

```

```

# 额外把噪声画出来（看不清噪声时非常有用）
plot(n, d, "k:", linewidth=1.5, alpha=0.8, label="d[n] (噪声)")

title("长度为 40 的受干扰序列（噪声幅度 ±0.6）")
ylabel("幅度")
legend(loc="best")
grid(true)

# (3) 中值滤波输出
subplot(4, 1, 3)
h3 = stem(n, y, markerfmt="ro", linefmt="r-", label="信号+噪声")
try
    c3 = (h3 isa AbstractVector && length(h3) == 1) ? h3[1] : h3
    if PyCall.pyhasattr(c3, "baseline")
        c3.baseline.set_visible(false)
    end
catch
end
plot(n, s+d, "k:", linewidth=2, alpha=0.8, label="信号+噪声")

title("信号+噪声")
xlabel("样本序号 n")
ylabel("幅度")
legend(loc="best")
grid(true)

# (2) 中值滤波输出
subplot(4, 1, 4)
h4 = stem(n, y, markerfmt="ro", linefmt="r-", label="y[n] (5点中值滤波输出)")
try
    c4 = (h4 isa AbstractVector && length(h4) == 1) ? h4[1] : h4
    if PyCall.pyhasattr(c4, "baseline")
        c4.baseline.set_visible(false)
    end
catch
end
plot(n, y, "k:", linewidth=2, alpha=0.8, label="y[n] (5点中值滤波输出)")

title("5 点中值滤波器输出")
xlabel("样本序号 n")
ylabel("幅度")
legend(loc="best")
grid(true)

try
    gcf().tight_layout()
catch
end

# 如果你只想严格按题意“分别绘制”且不叠加 s[n] 和 d[n]:
# - 把上面两个 subplot 里的 plot(n, s, ...) 和 plot(n, d, ...) 注释掉即可。

```

**18. 已知 16 点序列  $x[n]$  的 DFT 为:** 
$$X[k] = \begin{cases} k/16 & 0 \leq k \leq 15 \\ 0 & otherwise \end{cases}$$

**。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^4 + 0*z^3 + 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") # 保持纵横坐标比例一致, 使图看起来是圆的

```

**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 $x[n] = \cos(\pi n / 2), 0 \leq n \leq 15$ , 绘制序列 $x[n]$ 的 DFT 和 DTFT 的幅度。

```

using TyPlot
using FFTW

# =====
# 1. 信号定义
# =====
# 题目:  $x[n] = \cos(\pi * n / 2), 0 \leq n \leq 15$ 
N = 16
n = 0:N-1
x = cos.(pi .* n ./ 2)

# =====
# 2. 计算 DFT (离散傅里叶变换)
# =====
# DFT 计算的是离散频点  $k = 0, 1, \dots, N-1$ 
X_k = fft(x)
mag_X_k = abs(X_k)
k = 0:N-1

# =====
# 3. 计算 DTFT (离散时间傅里叶变换) - 近似
# =====
# DTFT 是连续频谱  $X(e^{j\omega})$ 。
# 为了在计算机中绘制, 我们对信号进行大量补零,
# 然后使用 FFT 计算出高密度的频域样本。

# 设置高分辨率点数 (比如 1024 或 2048)
K = 2048

# 构造补零后的信号: [原始信号, 0, 0, ...]
x_padded = [x; zeros(K - N)]

# 计算 FFT 并移位, 使零频率位于中心
X_dtft = fft(x_padded)
X_dtft_shifted = fftshift(X_dtft)
mag_X_dtft = abs(X_dtft_shifted)

# 生成归一化频率轴  $\omega / \pi$ 
# 范围: -1 到 1 (对应  $-\pi$  到  $\pi$ )
# 这样画图时, x轴为 0.5 就代表  $0.5\pi$  (即  $\pi/2$ )
w_normalized = range(-1, 1, length=K)

# =====
# 4. 绘图
# =====
figure("DFT and DTFT Analysis", figsize=(10, 8))

# 子图 1: 时域序列  $x[n]$ 
subplot(3, 1, 1)
# 修正: 移除了不兼容的 baselfmt 参数
stem(n, x, "b-o", label="x[n]")
title("时域序列  $x[n] = \cos(\pi n / 2)$ ")
xlabel("n")
ylabel("幅度")
grid(true)
xlim([-1, 16])

# 子图 2: DFT 幅度谱  $|X[k]|$  (N点)
subplot(3, 1, 2)
# 修正: 移除了不兼容的 baselfmt 参数
stem(k, mag_X_k, "r-o", label="|X[k]|")

```

```

title("DFT 幅度谱 |X[k]| (N=16)")
xlabel("频率索引 k")
ylabel("幅度")
grid(true)
xlim([0, 15])

# 解释: 对于 cos(pi*n/2), 频率是 1/4 fs。
# k = N * f = 16 * 0.25 = 4。所以 k=4 和 k=12 (16-4) 处有峰值。

# 子图 3: DTFT 幅度谱 |X(e^{j\omega})| (归一化频率)
subplot(3, 1, 3)
plot(w_normalized, mag_X_dtft, "g", linewidth=2, label="|X(e^{j\omega})|")
title("DTFT 幅度谱 (频率轴归一化: \times \pi rad/sample)")
xlabel("归一化频率 (\omega / \pi)") # 这样轴刻度 0.5 就代表 pi/2
ylabel("幅度")
grid(true)
xlim([-1, 1])

# 叠加显示 DFT 的点在 DTFT 上对应的位置 (为了展示 DFT 是 DTFT 的采样)
# DFT 的 k 对应模拟频率 w_k = 2*pi*k/N
# 归一化频率 w_norm = w_k / pi = 2*k/N
# 注意 fftshift 后的范围问题, 为了简单展示, 这里只画 DTFT 曲线
# 峰值应该出现在 +/- 0.5 处 (因为 pi/2 / pi = 0.5)

# 自动调整布局
# tight_layout()

```

## 23.已知 FIR 滤波器的系统函数为:

$$H(z) = 2.4 + 3.2z^{-1} + 1.5z^{-2} + 0.8z^{-3} + 1.4z^{-4} + 3.6z^{-5} + 5.2z^{-6}$$

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

```

```

    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  $0.35\pi$ , 阻带截止频率  $0.45\pi$ , 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("\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
#  $H(e^{j\omega}) = B(e^{j\omega}) / A(e^{j\omega})$ 
# 分子系数 (对应  $e^0, e^{-j\omega}, e^{-2j\omega}, \dots$ )
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 LinearAlgebra
using Printf

# =====
# 1) 求多项式根 (降幂系数) : c0*z^n + c1*z^(n-1) + ... + cn
# =====

function roots_desc(coeffs::AbstractVector{<:Real})
    n = length(coeffs) - 1
    @assert n >= 1 "多项式阶数必须≥1"
    c0 = coeffs[1]
```

```

@assert abs(c0) > 1e-12 "最高次系数不能为0"

c = coeffs ./ c0 # 归一化, 使最高次系数=1: z^n + c1 z^(n-1)+...+cn
C = zeros(ComplexF64, n, n)

# companion matrix:
# [ 0 0 ... 0 -cn
#   1 0 ... 0 -c(n-1)
#   0 1 ... 0 -c(n-2)
#   ...
#   0 0 ... 1 -c1 ]
for i in 2:n
    C[i, i-1] = 1.0
end
C[:, n] = -reverse(c[2:end]) # [-cn, ..., -c1]

return eigvals(C)
end

# =====
# 2) 共轭配对: 输出每个section的根(1个或2个)
# - 复根按 conjugate 配对 (保证二阶节实系数)
# - 实根两两配对, 若剩一个则形成一阶节
# =====
function pair_conjugates(r::Vector{ComplexF64}; tol=1e-6)
    r = copy(r)
    used = falses(length(r))
    pairs = Vector{Vector{ComplexF64}}()

    # 先处理复根 (找共轭)
    for i in eachindex(r)
        used[i] && continue
        if abs(imag(r[i])) > tol
            j = findfirst(j -> !used[j] && j != i && abs(r[j] - conj(r[i])) < 1e-4, eachindex(r))
            j == nothing && error("未找到共轭根: $(r[i]) (可能 tol 太小或根计算异常) ")
            push!(pairs, [r[i], r[j]])
            used[i] = true
            used[j] = true
        end
    end

    # 再处理实根
    real_roots = ComplexF64[]
    for i in eachindex(r)
        used[i] && continue
        if abs(imag(r[i])) <= tol
            push!(real_roots, ComplexF64(real(r[i]), 0.0))
            used[i] = true
        end
    end
    sort!(real_roots, by=x->real(x))

    i = 1
    while i <= length(real_roots)
        if i == length(real_roots)
            push!(pairs, [real_roots[i]]) # 一阶节
            i += 1
        else
            push!(pairs, [real_roots[i], real_roots[i+1]]) # 二阶节
            i += 2
        end
    end

    return pairs
end

# =====
# 3) 由 z 平面根构造 z^(n-1)=(q) 形式的节系数
# 若节含根 z1,z2:
# (1 - z1 q)(1 - z2 q) = 1 -(z1+z2)q + (z1 z2) q^2
# 若节含单根 z1:
# (1 - z1 q)
# =====
function section_coeff_from_zroots(zroots::Vector{ComplexF64})
    if length(zroots) == 2
        z1, z2 = zroots[1], zroots[2]
        b0 = 1.0
        b1 = -real(z1 + z2)
        b2 = real(z1 * z2)
        return [b0, b1, b2] # 升幂: b0 + b1 q + b2 q^2
    elseif length(zroots) == 1
        z1 = zroots[1]
    end
end

```

```

    b0 = 1.0
    b1 = -real(z1)
    return [b0, b1]    # 一阶: b0 + b1 q
else
    return [1.0]      # 空节 (不会用到)
end
end

# =====
# 4) 多项式卷积 (升幂系数: 常数项在前)
# =====
function conv(a::Vector{Float64}, b::Vector{Float64})
    y = zeros(Float64, length(a) + length(b) - 1)
    for i in eachindex(a)
        for j in eachindex(b)
            y[i+j-1] += a[i] * b[j]
        end
    end
    return y
end

# =====
# 5) 主程序: 题目系统
#  $H(z) = (2 + 5z^{-1} + z^{-2} - 3z^{-3} + 4z^{-4} + 6z^{-5}) / (1 + 3z^{-1} - 5z^{-2} + 2z^{-3} - 4z^{-4} + 3z^{-5})$ 
# =====
b = [2.0, 5.0, 1.0, -3.0, 4.0, 6.0] # B(q), q=z^{-1}, 升幂
a = [1.0, 3.0, -5.0, 2.0, -4.0, 3.0] # A(q), 升幂

# z 平面零点来自: z^5 B(1/z) 与 z^5 A(1/z)
# 它们的降幂系数正好是 [b0,b1,...,b5] 与 [a0,a1,...,a5] (对应 z^5 + ... + 常数)
Bz_desc = b          # 2 z^5 + 5 z^4 + ... + 6
Az_desc = a          # 1 z^5 + 3 z^4 + ... + 3

zzeros = roots_desc(Bz_desc)
zpoles = roots_desc(Az_desc)

println("z-plane zeros:")
println.(round.(zzeros, digits=6))
println("\nz-plane poles:")
println.(round.(zpoles, digits=6))

# 配对成级联节
zero_pairs = pair_conjugates(zzeros)
pole_pairs = pair_conjugates(zpoles)

# 级联节数量 (对齐: 不足的补1)
S = max(length(zero_pairs), length(pole_pairs))
while length(zero_pairs) < S
    push!(zero_pairs, ComplexF64[]) # 表示“无零点”的节 => 分子=1
end
while length(pole_pairs) < S
    push!(pole_pairs, ComplexF64[]) # 表示“无极点”的节 => 分母=1
end

# 总增益 (常数项比值)
k = b[1] / a[1]

println("\n=====")
println("级联型结构: H(z) = k * Π H_i(z)")
@printf("k = %.6f\n", k)
println("=====\\n")

# 构造并打印每个子系统
num_sections = Vector{Vector{Float64}}()
den_sections = Vector{Vector{Float64}}()

for i in 1:S
    bn = section_coeff_from_zroots(zero_pairs[i])
    ad = section_coeff_from_zroots(pole_pairs[i])

    push!(num_sections, bn)
    push!(den_sections, ad)

    # 打印表达式 (按 z^{-1})
    if length(bn) == 3
        @printf("H_%d(z) 分子: 1 %+.6f z^{-1} %+.6f z^{-2}\\n", i, bn[2], bn[3])
    elseif length(bn) == 2
        @printf("H_%d(z) 分子: 1 %+.6f z^{-1}\\n", i, bn[2])
    else
        @printf("H_%d(z) 分子: 1\\n", i)
    end
end

```

```

if length(ad) == 3
    @printf("    分母: 1 %+ .6f z^-1 %+ .6f z^-2\n\n", ad[2], ad[3])
elseif length(ad) == 2
    @printf("    分母: 1 %+ .6f z^-1\n\n", ad[2])
else
    @printf("    分母: 1\n\n")
end
end

function check_reconstruct(num_sections, den_sections, k, b, a)
    # 6) 重构校验: 把各节卷起来, 检查是否回到原 b,a
    b_rec = [1.0]
    a_rec = [1.0]
    for i in 1:length(num_sections)
        b_rec = conv(b_rec, num_sections[i])
        a_rec = conv(a_rec, den_sections[i])
    end
    b_rec .*= k

    while length(b_rec) < length(b)
        push!(b_rec, 0.0)
    end
    while length(a_rec) < length(a)
        push!(a_rec, 0.0)
    end

    println("==== 重构校验 (应与原系数一致) =====")
    println("原 b = ", b)
    println("重构b = ", round(b_rec, digits=6))
    println("原 a = ", a)
    println("重构a = ", round(a_rec, digits=6))

    err_b = maximum(abs.(b_rec[1:length(b)] .- b))
    err_a = maximum(abs.(a_rec[1:length(a)] .- a))
    @printf("max|Δb| = %.3e, max|Δa| = %.3e\n", err_b, err_a)
end

check_reconstruct(num_sections, den_sections, k, b, a)

```

**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  $0.55\pi$ , 阻带截止频率  $0.45\pi$ , 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 FFTW

# =====
# 0. 核心算法: 手动实现 Kaiser 窗
# =====
# 目的: 避免依赖 DSP.jl 和 SpecialFunctions.jl, 确保在 MWorks 中直接运行

function my_besseli0(x)
    s = 1.0; term = 1.0; x_half_sq = (x / 2)^2
    for k in 1:25
        term *= x_half_sq / (k^2); s += term
        if term < 1e-12 * s; break; end
    end
    return s
end

function my_kaiser(M::Int, beta::Float64)
    w = zeros(M); alpha = (M - 1) / 2.0; i0_beta = my_besseli0(beta)
    for n in 0:M-1
        val = beta * sqrt(1 - ((n - alpha) / alpha)^2)
        val = max(0.0, abs(val))
        w[n+1] = my_besseli0(val) / i0_beta
    end
    return w
end

# =====
# 1. 滤波器设计计算
# =====
# 指标
wp = 0.55 * pi # 通带截止

```

```

ws = 0.45 * pi # 阻带截止
delta = 0.04 # 波纹

# 中间参数
wc = (wp + ws) / 2
dw = abs(wp - ws)
A = -20 * log10(delta)

# 计算 Beta
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

# 计算阶数 N
N_est = (A - 8) / (2.285 * dw)
N = ceil(Int, N_est)
# 修正: 高通滤波器必须是偶数阶 (奇数长度 Type I)
if N % 2 != 0; N += 1; end
M = N + 1

println("-----")
println("设计结果:")
println(" 阻带衰减 A = $(round(A, digits=2)) dB")
println(" 滤波器阶数 N = $N (长度 M = $M)")
println("  Kaiser Beta = $(round(beta, digits=4))")
println("-----")

# 计算系数 h[n]
alpha_val = N / 2
n_seq = 0:N
h_ideal = zeros(M)
for i in 1:M
    val = n_seq[i]
    if val == alpha_val
        h_ideal[i] = 1.0 - (wc / pi)
    else
        m_val = val - alpha_val
        h_ideal[i] = -sin(wc * m_val) / (pi * m_val)
    end
end
# 加窗
h = h_ideal .* my_kaiser(M, beta)

# =====
# 2. 频率响应计算
# =====
K = 2048
H = fft([h; zeros(K - M)])
H_shifted = fftshift(H)

# 频率轴 (转换为数组以确保绘图兼容性)
freqs = collect(range(-1, 1, length=K))

# 幅度响应 (dB)
mag_H = abs.(H_shifted)
gain_dB = 20 .* log10.(mag_H ./ 1e-12)

# =====
# 3. 绘图结果
# =====
figure("High-pass Filter Design", figsize=(10, 8))

# 子图 1: 冲激响应
subplot(2, 1, 1)
stem(0:N, h, "b-o", label="h[n]")
title("FIR 高通滤波器冲激响应 h[n] (N=$N)")
xlabel("n")
ylabel("幅度")
grid(true)
legend()

# 子图 2: 增益响应
subplot(2, 1, 2)

# 1. 先画指标辅助线 (放在底层)
# 阻带/通带边界
plot([ws/pi, ws/pi], [-100, 20], "k--", label="阻带/通带边界")
plot([wp/pi, wp/pi], [-100, 20], "k--")
# 最小衰减线
plot([0, 1], [-A, -A], "k:", label="最小衰减指标")

# 2. 画增益曲线 (红色实线)

```

```

plot(freqs, gain_dB, "r", label="增益响应")

title("增益响应 (dB)")
xlabel("归一化频率 ( $\omega / \pi$ )")
ylabel("幅度 (dB)")
xlim(0, 1)    # 只显示 0 到 pi
ylim(-100, 10) # Y 轴范围: 显示从 -100dB 到 10dB

grid(true)
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  $H(z) = (1/6) * (1 + z^{-1} + \dots + z^{-5})$ 
# 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("\texttimes \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

# ===== 工具：求根（输入为“降幂”系数：c0*x^n + c1*x^(n-1)+...+cn)
function roots_desc(coeffs::AbstractVector{<:Real})
    n = length(coeffs) - 1
    @assert n >= 1
    lead = coeffs[1]
    @assert abs(lead) > 1e-12
    c = coeffs ./ lead # 归一化成首项为1

    C = zeros(ComplexF64, n, n)
    for i in 1:n-1
        C[i+1, i] = 1.0
    end
    C[:, n] = -c[2:end]
    eigvals(C)
end

# ===== 多项式求值（升幂：a0 + a1*q + ...）
polyval_asc(a, q) = sum(a[i] * q^(i-1) for i in 1:length(a))

# ===== 多项式导数（升幂系数 -> 升幂系数）
function polyder_asc(a::AbstractVector{<:Real})
    n = length(a) - 1
    n == 0 && return [0.0]
    d = zeros(Float64, n)
    for i in 2:length(a)
        d[i-1] = (i-1) * a[i]
    end
    d
end

# ===== MATLAB residuez 等价实现：b,a 为 z^-1(=q) 的升幂系数 [b0,b1,...]
# 返回 r,p,k, 使 H(z)=k + Σ r_i/(1 - p_i z^-1)
function residuez_like_matlab(b::Vector{Float64}, a::Vector{Float64})
    nb = length(b)-1
    na = length(a)-1
    @assert na >= 1

    # 这里只写够用版本：若 nb==na, 则 k 为常数（最高次项比值）
    # 你的题刚好 nb==na==5
    @assert nb == na "如需处理 nb≠na, 可再扩展做多项式长除法"
    k = b[end] / a[end] # 注意：最高次项比值（不是 b[1]/a[1]）
    b_rem = b .- k .* a # 余式（仍是升幂系数）

    # q=z^-1 域极点：A(q)=0 的根
    q_poles = roots_desc(reverse(a)) # reverse(a) 变成 q 的降幂系数
    p = 1.0 ./ q_poles # z 域极点

    # 留数：res_q = B_rem(q_i)/A'(q_i), 再换成 MATLAB 形式 r_i = -p_i * res_q
    a_der = polyder_asc(a)
    r = similar(p)
    for i in eachindex(q_poles)
        qi = q_poles[i]
        res_q = polyval_asc(b_rem, qi) / polyval_asc(a_der, qi)
        r[i] = -(p[i]) * res_q
    end

    # MATLAB 的 k 会把由换元产生的常数项也吸收进去；
    # 对于 r/(1-p z^-1)=r + (r p)/(z-p), 常数项是 r, 所以把 Σr 加到 k 里更接近 MATLAB 输出
    k = k + sum(real.(r)) # 系统系数全实，最终 k 应为实数
    return r, p, k
end

```

```

# ===== 题目系数
b = [2.0, 5.0, 1.0, -3.0, 4.0, 6.0]
a = [1.0, 3.0, -5.0, 2.0, -4.0, 3.0]

r, p, k = residuez_like_matlab(b, a)

println("Parallel Form I (complex):")
println("k = ", k)
for i in eachindex(p)
    println("r[$i] = ", r[i], ", p[$i] = ", p[i])
end

println("\nParallel Form II (real SOS):")
tol = 1e-6
used = falses(length(p))
for i in eachindex(p)
    used[i] && continue
    if abs(imag(p[i])) < tol
        println(" section: ", real(r[i]), " / (1 - ", real(p[i]), " z^-1)")
        used[i] = true
    else
        # 找共轭
        j = findfirst(j -> !used[j] && abs(p[j] - conj(p[i])) < 1e-5, eachindex(p))
        @assert j != nothing "未找到共轭极点, 请检查 tol"
        # 合并为二阶节 (实系数)
        b0 = 2*real(r[i])
        b1 = -2*real(r[i]*conj(p[i]))
        a1 = -2*real(p[i])
        a2 = abs(p[i])^2
        println(" section: (", b0, " + ", b1, " z^-1) / (1 + ", a1, " z^-1 + ", a2, " z^-2)")
        used[i] = true
        used[j] = true
    end
end
end

```

**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  $0.32\pi$  内幅度为  $0.6$ , 在频率范围  $0.35\pi$  到  $0.65\pi$  内幅度为  $0.2$ , 在频率范围  $0.68\pi$  到  $\pi$  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
    end

```

```

# 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

# coeffs = [a0, a1, ..., an] 表示 a0 + a1*x + ... + an*x^n
function poly_roots(coeffs::AbstractVector{<:Real})
    n = length(coeffs) - 1
    @assert n >= 1 "多项式阶数必须≥1"
    an = coeffs[end]
    @assert abs(an) > 1e-12 "最高次系数不能为0"

    c = reverse(coeffs) ./ an # [1, a_{n-1}/an, ..., a0/an]

    C = zeros(ComplexF64, n, n)
    C[1, :] .= -c[2:end]
    for i in 2:n
        C[i, i-1] = 1.0
    end
    eigvals(C)
end

# -----
# 题目: H(z)=B(z^-1)/A(z^-1)
# 乘 z^4 得到 z 平面多项式:
# Bz(z)=z^4 -0.2 z^3 + 0.5 z^2 + 2 z -0.6
# Az(z)=z^4 + 3.2 z^3 + 1.5 z^2 -0.8 z + 1.4
# -----

# 注意: poly_roots 需要的是 "升幂" 系数 [常数项, z^1, z^2, ..., z^4]
Bz = [-0.6, 2.0, 0.5, -0.2, 1.0]
Az = [1.4, -0.8, 1.5, 3.2, 1.0]

zzeros = poly_roots(Bz)
zpoles = poly_roots(Az)

println("Zeros (z-plane) = ", round.(zzeros, digits=4))
println("Poles (z-plane) = ", round.(zpoles, digits=4))

# ===== 绘图: 不使用 hold, 一次 plot 叠加三组数据 =====
theta = range(0, 2π, length=400)
ucx, ucy = cos.(theta), sin.(theta)

# 为了让零点/极点 "只有标记不连线", 用 NaN 打断折线
function nansep(v::AbstractVector{<:Real})
    out = Vector{Float64}(undef, 2length(v))
    out[1:2:end] .= v
    out[2:2:end] .= NaN
    out
end

```

```

zx = nansep(real.(zzeros)); zy = nansep(imag.(zzeros))
px = nansep(real.(zpoles)); py = nansep(imag.(zpoles))

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

# 一次画三组：单位圆(k--), 零点(bo)、极点(rx)
plot(ucx, ucy, "k--",
     zx, zy, "bo",
     px, py, "rx",
     linewidth=1, markersize=9)

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

# 范围
vals = vcat(real.(zzeros), imag.(zzeros), real.(zpoles), imag.(zpoles), [-1.0, 1.0])
m = maximum(abs.(vals)) + 0.5
xlim([-m, m]); ylim([-m, m])

# 图例 (如果这一行不兼容, 就删掉)
legend(["Unit Circle", "Zeros", "Poles"])

```

**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^N)
    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:  $x[n] = \sin(25\pi n/64)$ ,  $0 \leq n \leq 63$ , 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
using LinearAlgebra

# DFT (沿用你文件里的实现)
function my_dft(x)
    N = length(x)
    X = zeros{ComplexF64, N}
    for k in 0:N-1
        s = 0.0 + 0.0im
        for n in 0:N-1
            s += x[n+1] * exp(-2im*pi*k*n/N)
        end
        X[k+1] = s
    end
    X
end

N = 64
n = collect(0:N-1)
x = sin.(25*pi*n/64)

Xk = my_dft(x)

# 幅度 (不调用 abs) : |X| = sqrt(Re^2 + Im^2)
mag = sqrt(real(Xk).^2 .+ imag(Xk).^2)

k = collect(0:N-1)

figure("Q36: 64-point DFT of x[n]")

subplot(3,1,1)
stem(k, real(Xk), "b-", filled=true, markersize=4)
title("Re{X[k]}")
xlabel("k"); ylabel("Real")
grid("on")

subplot(3,1,2)
stem(k, imag(Xk), "b-", filled=true, markersize=4)
title("Im{X[k]}")
xlabel("k"); ylabel("Imag")
grid("on")

subplot(3,1,3)
stem(k, mag, "b-", filled=true, markersize=4)
title("Magnitude |X[k]| (computed by sqrt(Re^2+Im^2))")

```

```

xlabel("k"); ylabel("Magnitude")
grid("on")

# 标记真实“半个谱线”的位置: k0=12.5 和 N-k0=51.5
hold("on")
plot([12.5, 12.5], [0, maximum(mag)], "r--", linewidth=1, label="k=12.5")
plot([51.5, 51.5], [0, maximum(mag)], "r--", linewidth=1, label="k=51.5")
legend("on")
hold("off")

```

## 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(" -----")
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。")
```