1. FFT

function *thk* = *fft\_thk\_estimate*(*waveNum*, *R*, *n\_avg*, *waveN\_fit\_min*, *theta0\_deg*)

*% FFT\_THK\_ESTIMATE 通过傅里叶变换估算薄膜厚度*

*%   waveNum: 波数 (cm^-1)*

*%   R: 反射率*

*%   n\_avg: 外延层在该区域的平均折射率*

*%   waveN\_fit\_min: 用于FFT的最小波数*

*%   theta0\_deg: 入射角 (度)*

    if nargin < 5, theta0\_deg = 0; end

    theta0\_rad = theta0\_deg \* pi / 180;

    cos\_theta1 = *real*(*sqrt*(1 - (*sin*(theta0\_rad) / n\_avg)^2));

    filter = waveNum > waveN\_fit\_min;

    waveNum\_fft = *waveNum*(filter);

    R\_fft = *R*(filter);

*% 移除直流分量*

    R\_ac = R\_fft - *mean*(R\_fft);

*% 插值到均匀间隔的波数点*

    N = 2^*nextpow2*(8\**length*(waveNum\_fft));

    k\_uniform = *linspace*(*min*(waveNum\_fft), *max*(waveNum\_fft), N);

    r\_uniform = *interp1*(waveNum\_fft, R\_ac, k\_uniform, 'pchip', 'extrap');

*% 加窗并执行FFT*

    window = *hann*(N);

    r\_win = *r\_uniform*(:) .\* *window*(:);

    fft\_result = *fft*(r\_win);

    fft\_power = *abs*(*fft\_result*(1:N/2)).^2;

*% 计算厚度轴*

    dk = *mean*(*diff*(k\_uniform));

    thk\_axis = (0:N/2-1) \* 10000 / (2 \* n\_avg \* cos\_theta1 \* N \* dk);

*% 寻找主峰*

    search\_range = *find*(thk\_axis > 5 & thk\_axis < 200); *% 在合理范围内搜索*

    if isempty(search\_range)

        thk = 20;

        return;

    end

    [~, max\_idx\_in\_range] = *max*(*fft\_power*(search\_range));

    max\_idx\_rough = *search\_range*(max\_idx\_in\_range);

*% 能量重心矫正法*

    correctNum = 3;

    DatePower1 = 0;

    DatePower2 = 0;

    for i = -correctNum:correctNum

        idx = max\_idx\_rough + i;

        if idx >= 1 && idx <= *length*(fft\_power)

            power = *fft\_power*(idx);

            DatePower1 = DatePower1 + idx \* power;

            DatePower2 = DatePower2 + power;

        end

    end

    if DatePower2 > 0

        f\_corrected = DatePower1 / DatePower2;

    else

        f\_corrected = max\_idx\_rough;

    end

    thk = (f\_corrected - 1) \* 10000 / (2 \* n\_avg \* cos\_theta1 \* N \* dk);

end

2. clc;

clear;

close all;

file1.path = '附件3.xlsx';

file1.angle = 10;

file2.path = '附件4.xlsx';

file2.angle = 15;

config.waveN\_fit\_min = 400;

config.n1\_init = 3.48;

config.cauchyParam\_init = [3.42, 0.05];

config.selParam\_init = [10.6684293, 0.0030434748, 1.5413340, 0.301516485, 1.13475115, 1104];

config.k1Param\_A\_init = 1e-5;

config.k1Param\_B\_init = 2.0;

config.drudeParam\_init = [2000, 200];

config.epsilon\_inf = 11.7;

disp('一、FFT计算厚度初值');

data1 = readmatrix(file1.path);

thk\_init1 = fft\_thk\_estimate(data1(:,1), data1(:,2)/100, config.n1\_init, 2000, file1.angle);

data2 = readmatrix(file2.path);

thk\_init2 = fft\_thk\_estimate(data2(:,1), data2(:,2)/100, config.n1\_init, 2000, file2.angle);

config.thk\_init = mean([thk\_init1, thk\_init2]);

fprintf('10: %.2f μm\n', thk\_init1);

fprintf('15: %.2f μm\n', thk\_init2);

fprintf('平均厚度初值: %.2f μm\n', config.thk\_init);

fprintf('\n二、双光束干涉精细拟合\n')

fprintf('\n10:\n')

[res1] = process(data1, file1.angle, config, file1.path);

fprintf('\n15:\n')

[res2] = process(data2, file2.angle, config, file2.path);

analyze\_res(res1, res2, file1.angle, file2.angle);

function [res] = process(data, angle, config, output\_filepath)

waveNum = data(:, 1);

R = data(:, 2) / 100;

waveLen\_full = 10000 ./ waveNum;

filter = waveNum > config.waveN\_fit\_min;

R\_fit = R(filter);

waveLen\_fit = waveLen\_full(filter);

%% 柯西模型拟合

disp('柯西模型拟合');

x0\_cauchy = [config.thk\_init, config.cauchyParam\_init, config.k1Param\_A\_init, config.k1Param\_B\_init, config.drudeParam\_init];

lb\_cauchy = [config.thk\_init\*0.8, 3.35, 0, 0, 0, 100, 10];

ub\_cauchy = [config.thk\_init\*1.2, 3.6, 0.2, 1e-3, 4, 4000, 1000];

[x\_optimal\_cauchy, R\_squared\_cauchy] = global\_fit(x0\_cauchy, lb\_cauchy, ub\_cauchy, R\_fit, waveLen\_fit, angle, config.epsilon\_inf, 'cauchy');

cauchy\_res.thk = x\_optimal\_cauchy(1);

cauchy\_res.cauchyParam = x\_optimal\_cauchy(2:3);

cauchy\_res.k1Param = x\_optimal\_cauchy(4:5);

cauchy\_res.drudeParam = x\_optimal\_cauchy(6:7);

cauchy\_res.R\_squared = R\_squared\_cauchy;

cauchy\_res.model\_type = 'Cauchy';

cauchy\_res.n1\_complex = cal\_n\_cauchy(waveLen\_full, cauchy\_res.cauchyParam, cauchy\_res.k1Param);

cauchy\_res.n2\_complex\_full = cal\_n\_drude(waveNum, cauchy\_res.drudeParam, config.epsilon\_inf);

fprintf(' R^2: %.6f\n', R\_squared\_cauchy);

fprintf(' 厚度: %.2f μm\n', cauchy\_res.thk);

fprintf(' 柯西参数: A=%.3f, B=%.3f\n', cauchy\_res.cauchyParam(1), cauchy\_res.cauchyParam(2));

fprintf(' k1参数: A=%.2e, B=%.2f\n', cauchy\_res.k1Param(1), cauchy\_res.k1Param(2));

fprintf(' n2:Drude参数: ν\_p=%.1f, Γ=%.1f\n', cauchy\_res.drudeParam(1), cauchy\_res.drudeParam(2));

%% sel模型拟合

disp('sel模型拟合');

thk\_cauchy = x\_optimal\_cauchy(1);

k1\_cauchy = x\_optimal\_cauchy(4:5);

drude\_cauchy = x\_optimal\_cauchy(6:7);

x0\_sel = [thk\_cauchy, config.selParam\_init, k1\_cauchy, drude\_cauchy];

thk\_rf = 0.03;

lb\_thk = thk\_cauchy \* (1 - thk\_rf);

ub\_thk = thk\_cauchy \* (1 + thk\_rf);

sel\_rf = 0.2;

lb\_sel\_params = config.selParam\_init \* (1 - sel\_rf);

ub\_sel\_params = config.selParam\_init \* (1 + sel\_rf);

k1\_rf = 0.3;

lb\_k1 = k1\_cauchy \* (1 - k1\_rf);

ub\_k1 = k1\_cauchy \* (1 + k1\_rf);

drude\_rf = 0.1;

lb\_drude = drude\_cauchy \* (1 - drude\_rf);

ub\_drude = drude\_cauchy \* (1 + drude\_rf);

lb\_sel = [lb\_thk, lb\_sel\_params, lb\_k1, lb\_drude];

ub\_sel = [ub\_thk, ub\_sel\_params, ub\_k1, ub\_drude];

lb\_sel = max(lb\_sel, [config.thk\_init\*0.7, 0.1, 0.001, 0.0001, 0.0001, 0.1, 1, 0, 0, 50, 5]);

ub\_sel = min(ub\_sel, [config.thk\_init\*1.3, 20, 5, 10, 2, 15, 2000, 1e-3, 4, 5000, 1500]);

if lb\_sel(8) < 0

lb\_sel(8) = 0;

end

if lb\_sel(9) < 0

lb\_sel(9) = 0;

end

[x\_optimal\_sel, R\_squared\_sel] = global\_fit(x0\_sel, lb\_sel, ub\_sel, R\_fit, waveLen\_fit, angle, config.epsilon\_inf, 'sellmeier');

sel\_res.thk = x\_optimal\_sel(1);

sel\_res.selParam = x\_optimal\_sel(2:7);

sel\_res.k1Param = x\_optimal\_sel(8:9);

sel\_res.drudeParam = x\_optimal\_sel(10:11);

sel\_res.R\_squared = R\_squared\_sel;

sel\_res.model\_type = 'sel';

sel\_res.n1\_complex = cal\_n\_sellmeier(waveLen\_full, sel\_res.selParam, sel\_res.k1Param);

sel\_res.n2\_complex\_full = cal\_n\_drude(waveNum, sel\_res.drudeParam, config.epsilon\_inf);

...

fprintf(' R^2: %.6f\n', R\_squared\_sel);

fprintf(' 厚度: %.2f μm\n', sel\_res.thk);

fprintf(' 赛尔迈耶参数: %.4f, %.4f, %.4f, %.4f, %.4f, %.1f\n', sel\_res.selParam(1), sel\_res.selParam(2), sel\_res.selParam(3), sel\_res.selParam(4), sel\_res.selParam(5), sel\_res.selParam(6));

fprintf(' n2 Drude 参数: ν\_p= %.1f , Γ= %.1f \n', sel\_res.drudeParam(1), sel\_res.drudeParam(2));

...

%% 选择最优模型

disp('模型比较与选择');

fprintf(' 柯西模型 R^2 = %.6f\n', R\_squared\_cauchy);

fprintf(' sel模型 R^2 = %.6f\n', R\_squared\_sel);

if R\_squared\_sel > R\_squared\_cauchy

res = sel\_res;

res.selected\_model = 'sel';

fprintf(' 选择 sel 模型 (R^2提升: %.6f)\n', R\_squared\_sel - R\_squared\_cauchy);

else

res = cauchy\_res;

res.selected\_model = 'Cauchy';

fprintf(' 选择 柯西 模型 (R^2更高或相等)\n');

end

% 保存两个模型的结果以供后续分析

res.cauchy\_res = cauchy\_res;

res.sel\_res = sel\_res;

% 输出最终选择的模型参数

fprintf('\n最终选择: %s 模型\n', res.selected\_model);

fprintf(' 最终厚度: %.2f μm\n', res.thk);

fprintf(' 最终 R^2 = %.6f\n', res.R\_squared);

if strcmp(res.selected\_model, 'sel')

fprintf(' 外延层 n1 (在 6μm): %.3f + %.4fi\n', ...

real(interp1(waveLen\_full, res.n1\_complex, 6)), ...

imag(interp1(waveLen\_full, res.n1\_complex, 6)));

else

fprintf(' 外延层柯西参数: A=%.3f, B=%.3f\n', res.cauchyParam(1), res.cauchyParam(2));

fprintf(' 外延层 n1 (在 6μm): %.3f + %.4fi\n', ...

real(interp1(waveLen\_full, res.n1\_complex, 6)), ...

imag(interp1(waveLen\_full, res.n1\_complex, 6)));

end

fprintf(' k1参数: A=%.2e, B=%.2f\n', res.k1Param(1), res.k1Param(2));

fprintf(' Drude参数: ν\_p=%.1f , Γ=%.1f \n', res.drudeParam(1), res.drudeParam(2));

% 保存结果到Excel

try

theta0\_rad = angle \* pi / 180;

R\_fit\_full = compute\_R(res.thk, res.n1\_complex, res.n2\_complex\_full, waveNum, theta0\_rad, 1);

R\_fit\_per = R\_fit\_full \* 100;

writematrix(R\_fit\_per, output\_filepath, 'Sheet', 1, 'Range', 'C2');

fprintf(' 成功将拟合结果保存到: %s (使用%s模型)\n', output\_filepath, res.selected\_model);

catch ME

fprintf(' 保存Excel文件时出错: %s\n', ME.message);

end

figure;

% 柯西

subplot(1,2,1);

R\_fit\_cauchy = compute\_R(cauchy\_res.thk, cauchy\_res.n1\_complex, cauchy\_res.n2\_complex\_full, waveNum, angle \* pi / 180, 1);

plot(waveNum(2:end), R(2:end)\*100, '-', 'Color', '#d74f44', 'LineWidth', 1.5, 'DisplayName', '实验数据');

hold on;

plot(waveNum, R\_fit\_cauchy\*100, '-', 'Color', '#008ede', 'LineWidth', 1.3, 'DisplayName', sprintf('柯西模型 (R^2=%.4f)', cauchy\_res.R\_squared));

xlabel('波数 (cm^{-1})');

ylabel('反射率 (%)');

legend('Location', 'best');

grid on;

% sel

subplot(1,2,2);

R\_fit\_sel = compute\_R(sel\_res.thk, sel\_res.n1\_complex, sel\_res.n2\_complex\_full, waveNum, angle \* pi / 180, 1);

plot(waveNum(2:end), R(2:end)\*100, '-', 'Color', '#d74f44', 'LineWidth', 1.5, 'DisplayName', '实验数据');

hold on;

plot(waveNum, R\_fit\_sel\*100, '-', 'Color', '#3f8819', 'LineWidth', 1.3, 'DisplayName', sprintf('sel模型 (R^2=%.4f)', sel\_res.R\_squared));

xlabel('波数 (cm^{-1})');

ylabel('反射率 (%)');

legend('Location', 'best');

grid on;

% 绘制最优模型的光学常数

plot\_optical\_constants(waveLen\_full, waveNum, res.n1\_complex, sprintf('外延层 (%s)', res.selected\_model));

plot\_optical\_constants(waveLen\_full, waveNum, res.n2\_complex\_full, '衬底 (Drude)');

end

function *R* = *cal\_R\_db*(*r01\_s*, *r12\_s*, *r01\_p*, *r12\_p*, *exp\_term*)

*% CALCULATE\_R\_DOUBLE\_BEAM 双光束干涉模型*

    r\_s\_total = r01\_s + r12\_s .\* exp\_term;

    r\_p\_total = r01\_p + r12\_p .\* exp\_term;

    R\_s = *abs*(r\_s\_total).^2;

    R\_p = *abs*(r\_p\_total).^2;

    R = *real*((R\_s + R\_p) / 2);

end

三、

clc;

clear;

*close* all;

file1.path = '附件1.xlsx';

file1.angle = 10; *% 入射角 (度)*

file2.path = '附件2.xlsx';

file2.angle = 15; *% 入射角 (度)*

config.waveN\_fit\_min = 1500; *% 拟合起始波数 (cm-1)*

config.n1\_init = 2.58;       *% 外延层折射率初值 (用于FFT估算厚度)*

config.n2\_real\_init = 2.55;  *% 衬底折射率初值*

config.k1\_A\_init = 0.001;    *% 外延层消光系数参数A初值*

config.k1\_B\_init = 2.0;      *% 外延层消光系数参数B初值*

*% Sellmeier模型 B1, B2, B3, C1, C2, C3 参数初值*

config.selParam\_init = [5.5, 0.2, 0.05, 0.027, 100, 0.01];

*disp*('FFT计算厚度初值');

data1 = *readmatrix*(file1.path);

thk\_init1 = *fft\_thk\_estimate*(*data1*(:,1), *data1*(:,2)/100, config.n1\_init, config.waveN\_fit\_min, file1.angle);

data2 = *readmatrix*(file2.path);

thk\_init2 = *fft\_thk\_estimate*(*data2*(:,1), *data2*(:,2)/100, config.n1\_init, config.waveN\_fit\_min, file2.angle);

config.thk\_init = *mean*([thk\_init1, thk\_init2]);

*fprintf*('文件1 (10°) FFT估算厚度: %.2f μm\n', thk\_init1);

*fprintf*('文件2 (15°) FFT估算厚度: %.2f μm\n', thk\_init2);

*fprintf*('<strong>平均厚度初值: %.2f μm</strong>\n', config.thk\_init);

*disp*(' ');

*disp*('10：');

[res1] = *process*(data1, file1.angle, config, file1.path);

*disp*('15：');

[res2] = *process*(data2, file2.angle, config, file2.path);

*disp*(' ');

*disp*('分析：');

*analyze\_res*(res1, res2, file1.angle, file2.angle);

*%%* *数据处理和拟合主函数*

function [*res*] = *process*(*data*, *angle*, *config*, *output\_filepath*)

    epsilon= 11.7;

    waveNum = *data*(:, 1);       *% 波数 (cm-1)*

    R = *data*(:, 2) / 100;       *% 反射率 (0-1)*

    waveLen\_full = 10000 ./ waveNum; *% 波长 (μm)*

    filter = waveNum >= config.waveN\_fit\_min;

    waveNum\_fit = *waveNum*(filter);

    waveLen\_fit = 10000 ./ waveNum\_fit;

    R\_fit = *R*(filter);

*% 厚度, 衬底n2, Sellmeier(6), k参数(2)*

    x0 = [config.thk\_init, config.n2\_real\_init, config.selParam\_init, config.k1\_A\_init, config.k1\_B\_init];

    lb = [config.thk\_init\*0.8, 2.0, 0.01, 0.001, 0.0001, 0.0001, 0.1, 0.001, 0, 0];

    ub = [config.thk\_init\*1.2, 3.5, 20, 10, 5, 2, 150, 20, 0.01, 4];

    model\_type = 'real\_substrate';

    [x\_optimal, R\_squared\_fit] = *global\_fit*(x0, lb, ub,  R\_fit,waveLen\_fit, angle, epsilon ,model\_type);

    res.thk = *x\_optimal*(1);

    res.n2 = *x\_optimal*(2);

    res.selParam = *x\_optimal*(3:8);

    res.k1Param = *x\_optimal*(9:10);

    res.n1\_complex\_full = *cal\_n\_sellmeier*(waveLen\_full, res.selParam, res.k1Param);

*fprintf*('    最终厚度: %.2f μm\n', res.thk);

*fprintf*('    最终外延层 n1 (在 6μm): %.3f + %.4fi\n', *real*(*interp1*(waveLen\_full, res.n1\_complex\_full, 6)), *imag*(*interp1*(waveLen\_full, res.n1\_complex\_full, 6)));

*fprintf*('    最终衬底 n2: %.3f\n', res.n2);

*fprintf*('    在拟合区域的拟合优度 R?: %.4f\n', R\_squared\_fit);

    theta0\_rad = angle \* pi / 180;

    R\_fit\_full = *compute\_R*(res.thk, res.n1\_complex\_full, res.n2, waveNum, theta0\_rad,1);

    R\_fit\_per = R\_fit\_full \* 100;

    header = {'拟合反射率'};

*writecell*(header, output\_filepath, 'Sheet', 1, 'Range', 'C1');

*writematrix*(R\_fit\_per, output\_filepath, 'Sheet', 1, 'Range', 'C2');

*plot\_fit\_res*(waveNum, R, R\_fit\_full, angle, config.waveN\_fit\_min);

*plot\_optical\_constants*(waveLen\_full, waveNum, res.n1\_complex\_full, '外延层 (Sellmeier)');

end

function [*x\_optimal*, *R\_squared*] = *global\_fit*(*x0*, *lb*, *ub*, *R\_fit*, *waveLen\_fit*, *angle*, *epsilon\_inf*, *model\_type*)

    waveNum\_fit = 10000 ./ waveLen\_fit;

    theta0\_rad = angle \* pi / 180;

    if *strcmp*(model\_type, 'cauchy')

        model\_func = @(*x*, *k*) *model\_R\_cauchy*(x, k, waveLen\_fit, theta0\_rad, epsilon\_inf);

    elseif *strcmp*(model\_type, 'sellmeier')

        model\_func = @(*x*, *k*) *model\_R\_sellmeier*(x, k, waveLen\_fit, theta0\_rad, epsilon\_inf);

    elseif *strcmp*(model\_type, 'real\_substrate')

        model\_func = @(*x*, *k*) *model\_R\_real\_substrate*(x, k, waveLen\_fit, theta0\_rad);

    end

    options = *optimoptions*('lsqcurvefit', 'Display', 'off', 'MaxIterations', 1000, 'FunctionTolerance', 1e-9, 'StepTolerance', 1e-10, 'Algorithm', 'trust-region-reflective', 'UseParallel', true);

    [x\_optimal, resnorm] = *lsqcurvefit*(model\_func, x0, waveNum\_fit, R\_fit, lb, ub, options);

    SS\_tot = *sum*((R\_fit - *mean*(R\_fit)).^2);

    R\_squared = 1 - resnorm / SS\_tot;

end

function *R* = *model\_R\_sellmeier*(*params*, ~, *waveLen*, *theta0*, *epsilon\_inf*)

    thk = *params*(1);

    selParam = *params*(2:7);

    k1Param = *params*(8:9);

    drudeParam = *params*(10:11);

    waveNum\_model = 10000 ./ waveLen;

    n1\_complex = *cal\_n\_sellmeier*(waveLen, selParam, k1Param);

    n2\_complex = *cal\_n\_drude*(waveNum\_model, drudeParam, epsilon\_inf);

    R = *compute\_R*(thk, n1\_complex, n2\_complex, waveNum\_model, theta0,1);

end

function *R* = *model\_R\_real\_substrate*(*params*, ~, *waveLen*, *theta0*)

    thk = *params*(1);

    n2 = *params*(2);

    selParam = *params*(3:8);

    k1Param = *params*(9:10);

    waveNum\_model = 10000 ./ waveLen;

    n1\_complex = *cal\_n\_sellmeier*(waveLen, selParam, k1Param);

    R = *compute\_R*(thk, n1\_complex, n2, waveNum\_model, theta0, 1);

end

function *R* = *model\_R\_cauchy*(*params*, ~, *waveLen*, *theta0*, *epsilon\_inf*)

    thk = *params*(1);

    cauchyParam = *params*(2:3);

    k1Param = *params*(4:5);

    drudeParam = *params*(6:7);

    waveNum\_model = 10000 ./ waveLen;

    n1\_complex = *cal\_n\_cauchy*(waveLen, cauchyParam, k1Param);

    n2\_complex = *cal\_n\_drude*(waveNum\_model, drudeParam, epsilon\_inf);

    R = *compute\_R*(thk, n1\_complex, n2\_complex, waveNum\_model, theta0,1);

end

function *R* = *compute\_R*(*thk*, *n1\_complex*, *n2\_complex*, *waveNum*, *theta0*, *opt*)

*% compute\_R 计算单层膜的反射率*

*%   thk: 膜厚 (μm)*

*%   n1\_complex: 外延层复折射率 (n+ik)*

*%   n2\_complex: 衬底复折射率 (n+ik)*

*%   waveNum: 波数 (cm^-1)*

*%   theta0: 入射角 (弧度)*

*%   opt: 选项*

*%        0 - 强制使用双光束模型*

*%        1 - 强制使用多光束模型*

*%        2 - 自适应选择（推荐）*

    n0 = 1.0;

    n1 = *n1\_complex*(:);

    n2 = *n2\_complex*(:);

    waveNum = *waveNum*(:);

    sin\_theta1 = n0 \* *sin*(theta0) ./ n1;

    cos\_theta1 = *sqrt*(1 - sin\_theta1.^2);

    sin\_theta2 = n0 \* *sin*(theta0) ./ n2;

    cos\_theta2 = *sqrt*(1 - sin\_theta2.^2);

    r01\_s = (n0\**cos*(theta0) - n1.\*cos\_theta1) ./ (n0\**cos*(theta0) + n1.\*cos\_theta1);

    r12\_s = (n1.\*cos\_theta1 - n2.\*cos\_theta2) ./ (n1.\*cos\_theta1 + n2.\*cos\_theta2);

    r01\_p = (n1\**cos*(theta0) - n0\*cos\_theta1) ./ (n1\**cos*(theta0) + n0.\*cos\_theta1);

    r12\_p = (n2.\*cos\_theta1 - n1.\*cos\_theta2) ./ (n2.\*cos\_theta1 + n1.\*cos\_theta2);

    delta = 4 \* pi \* n1 .\* thk .\* cos\_theta1 .\* waveNum / 10000;

    exp\_term = *exp*(1i\*delta);

    if opt == 0

*% 强制双光束*

        R = *cal\_R\_db*(r01\_s, r12\_s, r01\_p, r12\_p, exp\_term);

    elseif opt == 1

*% 强制多光束*

        R = *cal\_R\_mul*(r01\_s, r12\_s, r01\_p, r12\_p, exp\_term);

    elseif opt == 2

*% 自适应选择模型*

        epsilon\_threshold = 0.01;

        [use\_multi\_beam, ~] = *check*(thk, n1\_complex, n2\_complex, waveNum, theta0, epsilon\_threshold);

        R = *zeros*(*size*(waveNum));

*% 分别处理需要不同模型的波长点*

        if any(use\_multi\_beam)

            idx = use\_multi\_beam;

*R*(idx) = *cal\_R\_mul*(*r01\_s*(idx), *r12\_s*(idx), *r01\_p*(idx), *r12\_p*(idx), *exp\_term*(idx));

        end

        if any(~use\_multi\_beam)

            idx = ~use\_multi\_beam;

*R*(idx) = *cal\_R\_db*(*r01\_s*(idx), *r12\_s*(idx), *r01\_p*(idx), *r12\_p*(idx), *exp\_term*(idx));

        end

    elseif opt >= 0.005 && opt <= 0.05

*% 使用opt作为自适应阈值*

        epsilon\_threshold = opt;

        [use\_multi\_beam, ~] = *check*(thk, n1\_complex, n2\_complex, waveNum, theta0, epsilon\_threshold);

        R = *zeros*(*size*(waveNum));

        if any(use\_multi\_beam)

            idx = use\_multi\_beam;

*R*(idx) = *cal\_R\_mul*(*r01\_s*(idx), *r12\_s*(idx), *r01\_p*(idx), *r12\_p*(idx), *exp\_term*(idx));

        end

        if any(~use\_multi\_beam)

            idx = ~use\_multi\_beam;

*R*(idx) = *cal\_R\_db*(*r01\_s*(idx), *r12\_s*(idx), *r01\_p*(idx), *r12\_p*(idx), *exp\_term*(idx));

        end

    else

*% 默认使用多光束*

        R = *cal\_R\_mul*(r01\_s, r12\_s, r01\_p, r12\_p, exp\_term);

    end

end