Gerchberg Saxton algorithm

Let:

FT – forward Fourier transform

IFT – inverse Fourier transform

i – the imaginary unit, √−1 (square root of −1)

exp – exponential function (exp(x) = ex)

Target and Source be the Target and Source Amplitude planes respectively

A, B, C & D be complex planes with the same dimension as Target and Source

Amplitude – Amplitude-extracting function:

e.g. for complex z = x + iy, amplitude(z) = sqrt(x·x + y·y)

for real x, amplitude(x) = x

Phase – Phase extracting function:

e.g. Phase(z) = arctan(y / x)

end

Let

algorithm Gerchberg–Saxton(Source, Target, Retrieved_Phase) is

A := IFT(Target) while error criterion is not satisfied

B := Amplitude(Source) × exp(i × Phase(A))

C := FT(B)

D := Amplitude(Target) × exp(i × Phase(C))

A := IFT(D)

end while

Retrieved_Phase = Phase(A)

```
close all
clear
```

Load image and create mask

```
surface_image = importdata("zernike6th.mat"); % rads of optical phase;
mask = ones(size(surface_image));
mask(isnan(surface_image)) = 0;

surface_image(isnan(surface_image)) = 0 ;
figure(1)
imagesc(surface_image);
```

```
title('Surface deformation')
cb = colorbar;
ylabel(cb,'(rads)','FontSize',14);
xlabel('pixels')
ylabel('pixels')

lambda = 550; % wavelenth 550nm
k = 2*pi/lambda;  % wavenumber

scalarMultiplier = 2; % increase PV from surface image (tested from 1-10)
surface_image = scalarMultiplier*surface_image;

% peak-to-valley
PV = (max(surface_image(:))-min(surface_image(:)))/k;
% residual error
RMS_PV = std(surface_image(mask>0))/k;

% start by fitting the image in ratio 50%
N = length(surface_image);
tmp_image = zeros(2*N, 2*N);
tmp_image(N/2+1:N+N/2, N/2+1:N+N/2) = surface_image;
surface_image = tmp_image;

figure(2)
imagesc(surface_image);
title('Surface deformation')
cb = colorbar;
ylabel(cb,'(rads)','FontSize',14);
xlabel('pixels')
ylabel('pixels')

surface_image_nm = surface_image/k;
figure(3)
imagesc(surface_image_nm);
cb = colorbar; ylabel(cb,'(nm)','FontSize',14);
title("Surface deformation")
xlabel('pixels')
ylabel('pixels')
```

Define parameters for G-S Algorithm

```
x = linspace(-1,1,N);
y = linspace(-1,1,N);
[X,Y] = meshgrid(x,y);
x0 = 0; % center
y0 = 0; % center
sigma = 2;
res = ((X-x0).^2 + (Y-y0).^2)./(2*sigma^2);
% create Source with the original image size and then center it in (2Nx2N)
Source = exp(-res).*mask;  % amplitude profile mirror surface

% fitting the image in ratio 50%
tmp_image = zeros(2*N, 2*N);
tmp_image(N/2+1:N+N/2, N/2+1:N+N/2) = Source;
```

```
Source = tmp_image;

actSource = abs( Source ).* exp(1i*surface_image); % complex amplitude field;
actCam = abs( fftshift( fft2( fftshift( actSource ) ) ) ).^2; % Simulation camera measure ; FF

A = abs(Source) ;  % inital A

% make mask 2N x 2N
tmp_image = zeros(2*N, 2*N);
tmp_image(N/2+1:N+N/2, N/2+1:N+N/2) = mask;
mask = tmp_image;

unwrapStruct = mksprecon(find(mask>0), size(surface_image));
```

Gerchberg–Saxton algorithm

```
error = [];
iteration_num = 500;
flip_cmd = 0; % in case we need to rotate the image
i = 1;
figure

while i < iteration_num   % convergence criteria or total iterations

    [~, unwrappedPhaseA] = spunwrap(angle(A), unwrapStruct);
    B = abs(Source) .* exp(1i*(unwrappedPhaseA));   %  Amplitude(Source) × exp(i × Phase(A))
    C = fftshift(fft2(fftshift(B)));  %  FT(B)
    D = abs ( sqrt(actCam) ) .* exp( 1i*angle(C) ); %  Amplitude(Target) × exp(i × Phase(C))
    A = fftshift(ifft2(fftshift(D)));  %  IFT(D)

    if flip_cmd == 1
        tmp_phaseA = flipud(fliplr(-unwrappedPhaseA));
    else
        tmp_phaseA = unwrappedPhaseA;
    end
    error = [error; (std( tmp_phaseA(mask>0) - surface_image(mask>0) ))];  % rms wavefront erro

    if i == 20 % verify if the error is growing to decide to flip the image
        p = polyfit(3:i, error(3:i),1);
        err_tmp = diff(error(3:end));
        if p(1) > 0.0009 % err_tmp > 0
            flip_cmd = 1;
            i = 1;
            error = error(1);
        end
    end

    subplot(3,1,1)
    imagesc(tmp_phaseA); cb = colorbar; ylabel(cb,'(rads)','FontSize',14); % Present current pa
    title(sprintf('phase A, iteration %d',i));
    xlabel('pixels')
    ylabel('pixels')
    subplot(3,1,2)
```

```
    imagesc(abs(C)); colorbar
    subplot(3,1,3)
    imagesc(sqrt(actCam)); colorbar

    drawnow

    if i > 31  % check if the error remains constant or if is less than a threshold
        err_tmp = diff(error(end-30:end));
        p = polyfit(i-30:i, error(i-30:i),1);
        if abs(error(end-10:end)) < 0.0001
            fprintf("Abs error < 0.0001,  %4.4f\n", err_tmp(end));
            break;
        end
        if (abs(p(1)) <.0001)
            fprintf("Error trending < 0.0001, %4.4f\n", p(1));
            break
        end
    end

    i = i+1;

end
```

Plots

```
figure
i = 1:1:length(error);
plot(i,(error'));
title('RMS wavefront error'); grid on
xlabel('iteration')
ylabel('rads')

% divide by 2pi; RMS wavefront error; y label waves
error = error/(2*pi);
figure
plot(i,(error'));
title('RMS wavefront error'); grid on
xlabel('iteration')
ylabel('waves')

error = error * lambda;
figure
plot(i,(error'));
title('RMS wavefront error'); grid on
xlabel('iteration')
ylabel('nm of OPD')

% Create phase unwrapping structure
[~, unwrappedPhaseA] = spunwrap(angle(A), unwrapStruct);
phase = unwrappedPhaseA; % OPD optical path difference
if flip_cmd == 1
    fixed_phase = flipud(fliplr(-phase));
else
```

```
    fixed_phase = phase;
end

fixed_nm = fixed_phase/k; % k = 2*pi/lambda

figure
imagesc(fixed_phase);
title("Phase")
cb = colorbar; ylabel(cb,'(rads)','FontSize',14);
xlabel('pixels')
ylabel('pixels')

figure
imagesc(fixed_nm);
cb = colorbar; ylabel(cb,'(nm)','FontSize',14);
title("OPD")
xlabel('pixels')
ylabel('pixels')

% compare original vs Phase
err = surface_image - fixed_phase;
figure;
imagesc(err);
cb = colorbar; ylabel(cb,'(rads)','FontSize',14);
title("Error")
xlabel('pixels')
ylabel('pixels')


err_nm = surface_image_nm - fixed_nm;
figure
imagesc(err_nm);
cb = colorbar; ylabel(cb,'(nm)','FontSize',14);
title("Error")
xlabel('pixels')
ylabel('pixels')

fprintf("PV (nm): %4.4f\n", PV);
fprintf("RMS PV(nm): %4.4f\n", RMS_PV); % compare
fprintf("Final error (nm): %4.4f\n", error(end)); % compare
fprintf("Scalar: %d\n", scalarMultiplier);
fprintf("Iterations: %d\n", i(end)); %iteration_num);
```