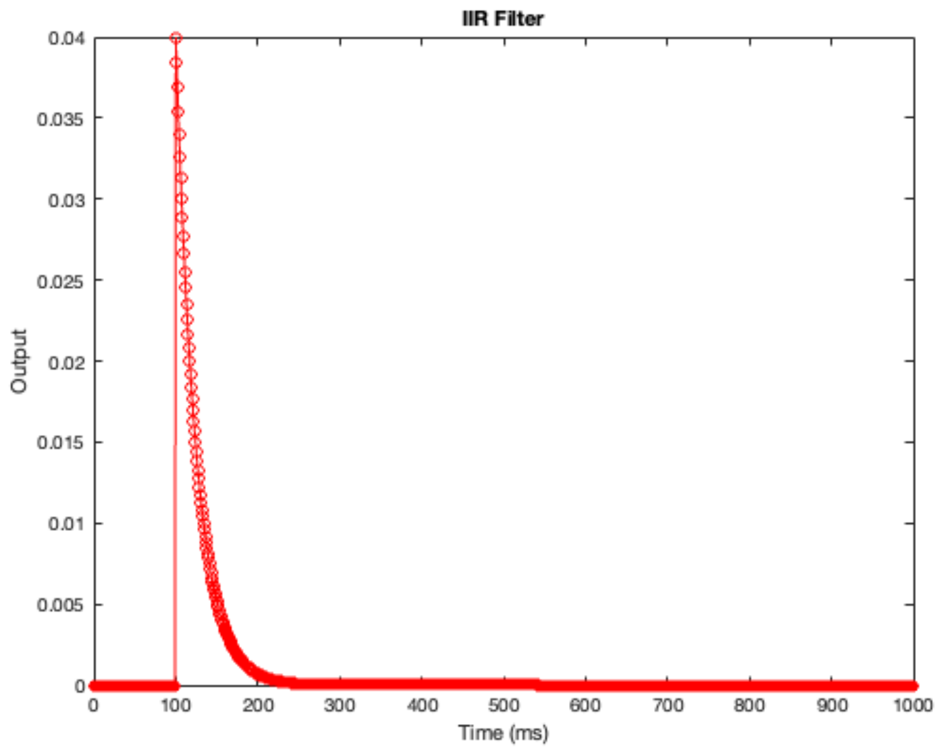
Contents

```
clear; close all; clc;
```

The IIR Filter is defined using an ODE. Solving the ODE provides a solution for the output y given an input x. The iir_filter function is defined here that solves the ODE.

```
tic
deltaT = 1; % ms
duration = 1000; % ms
t = 0:deltaT:duration-deltaT;
x = zeros(size(t));
x(100) = 1;
tau = 25; % ms

y1 = iir_filter(x, t, deltaT, tau);
figure();
plot(t, y1, 'ro-');
xlabel('Time (ms)')
ylabel('Output')
title('IIR Filter')
```

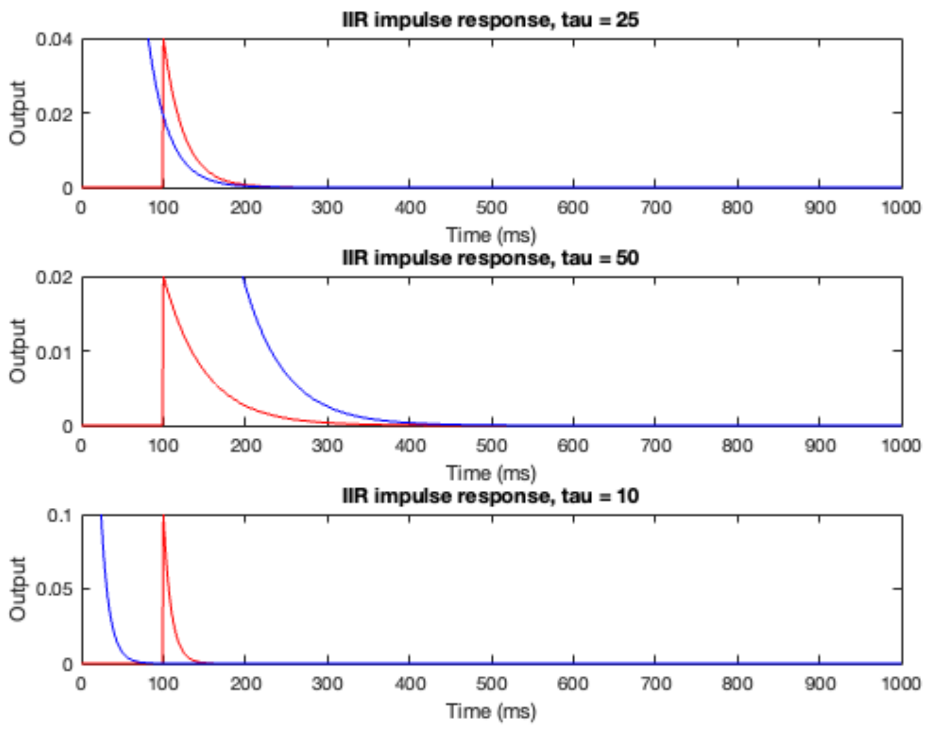


a)

Impulse response

Running the iir_filter for different time-constants and plotting the output along-with an exponential function:

```
deltaT = 1; % ms
duration = 1000; % ms
t = 0:deltaT:duration-deltaT;
% The impulse at time-step = 100 ms
x = zeros(size(t));
x(100) = 1;
taus = [25, 50, 10]; % ms

figure();
for ll = 1:length(taus)
    tau = taus(ll);
    y1 = iir_filter(x, t, deltaT, tau);

    subplot(3, 1, ll)
    plot(t, y1, 'r-');
    hold on;

    exponential = exp(-(t./tau)); % Exponential function
    plot(exponential, 'b');

    xlabel('Time (ms)')
    ylabel('Output')
    ylim([0, max(y1)])
    title(['IIR impulse response, tau = ', num2str(tau)])
end
```



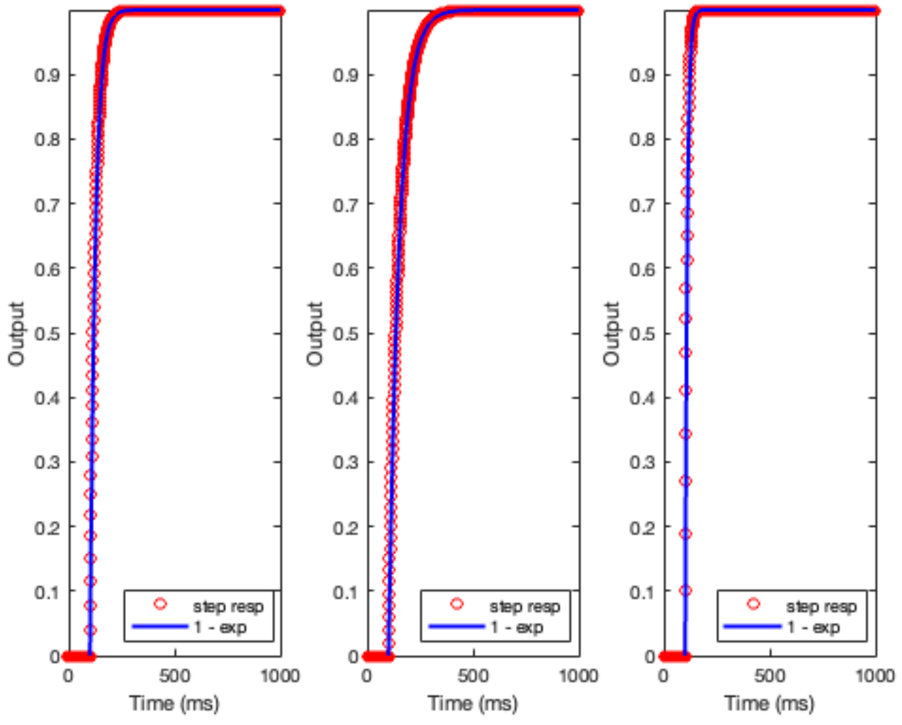
The exponential function $e^{-t/\tau}$ seems to fit the output. However, it is translated along x-direction and can be re-adjusted to account for the position of the impulse. This suggests that the solution of the ODE is of the form $e^{-t/\tau}$ which is trivial.

b)

Step response

Here I am using a trivial step function where the output of the function is 0 and becomes 1 after a certain time-step. For consistency, the change is kept at 100 ms.

```
deltaT = 1; % ms
duration = 1000; % ms
t = 0:deltaT:duration-deltaT;
% Step function at time-step = 100 ms
x = zeros(size(t));
x(100:1000) = 1;
taus = [25, 50, 10]; % ms

figure();
for ll = 1:length(taus)
    tau = taus(ll);
    y1 = iir_filter(x, t, deltaT, tau);

    subplot(1, 3, ll)
    plot(t, y1, 'ro', 'DisplayName', 'step resp');
    hold on;

    t_new = t-100; % translating time-axis to account for the position of step function start
    exponential = 1 - exp(-(t_new./tau)); % Fitting 1 - exp
    plot(exponential, 'b', 'LineWidth', 2, 'DisplayName', '1 - exp');

    xlabel('Time (ms)')
    ylabel('Output')
    ylim([0, max(y1)])
    legend('Location', 'southeast')
end
```



From the graphs, we can see that the output of the IIR filter to the step response is indeed of the form $1 - e^{-t/\tau}$

c)

Sinusoidal response

Passing a sinusoid through an IIR filter returns a sinusoid of the same frequency since this is a linear system. The amplitude and phase of the output can be computed from the fft of the ODE. The ODE for the IIR filter is:

$$\tau \frac{dy(t)}{dt} = -y(t) + x(t)$$

Computing fft on both sides of the equation, we get:

$$\tau \hat{d}(\omega)\hat{y}(\omega) = -\hat{y}(\omega) + \hat{x}(\omega)$$

Solving for $\hat{y}(\omega)$ gives:

$$\hat{y}(\omega) = \frac{\hat{x}(\omega)}{1 + \tau \hat{d}(\omega)}$$

Let

$$\hat{h}(\omega) = \frac{1}{1 + \tau \hat{d}(\omega)}$$

where $\hat{h}(\omega)$ is the effective frequencye response of the linear system. Therefore,

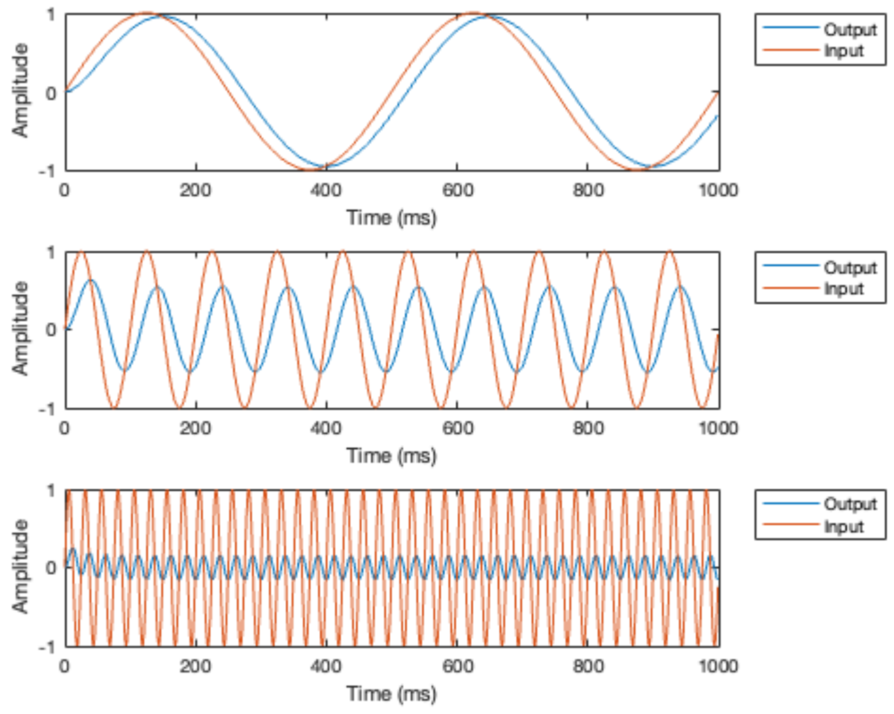
$$\hat{y}(\omega) = \hat{h}(\omega)\hat{x}(\omega)$$

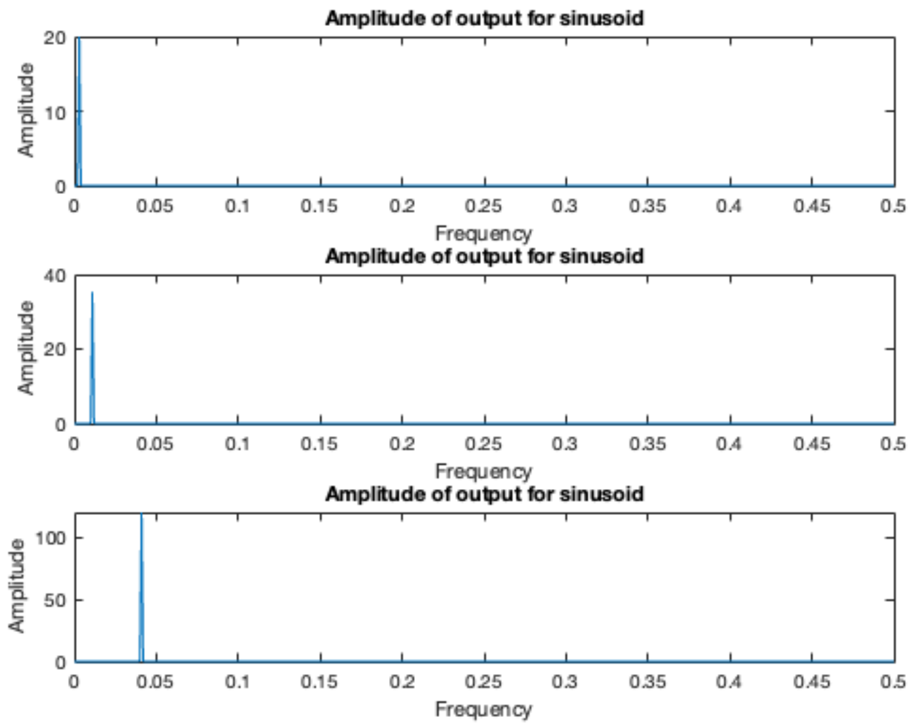
The amplitude and phase can then be computed using the equation:

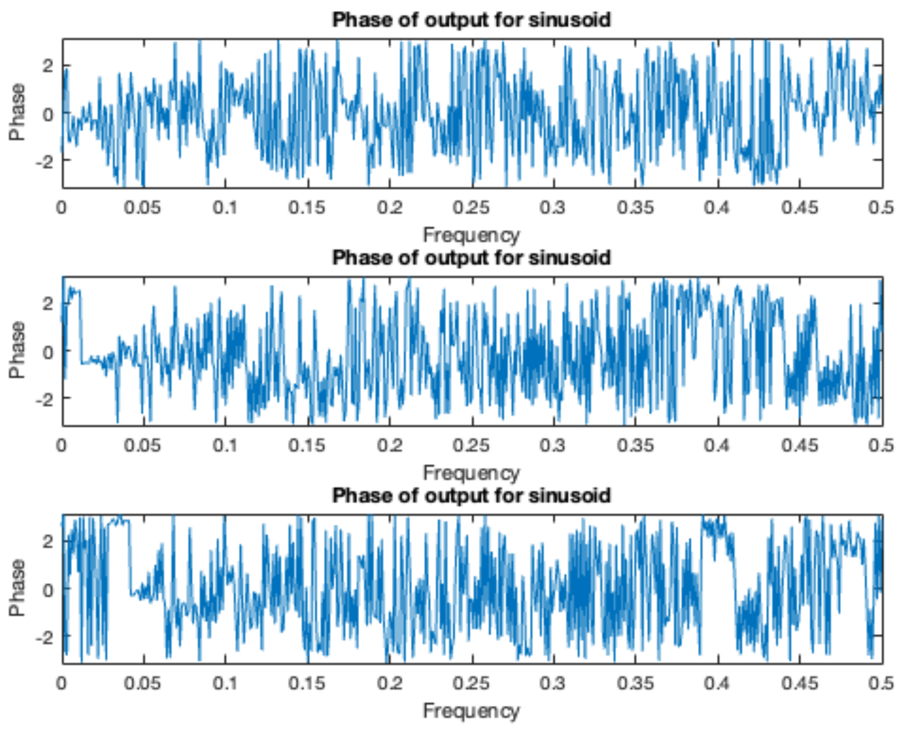
$$amp[\hat{y}(\omega)] = amp[\hat{h}(\omega)] \times amp[\hat{x}(\omega)]$$

$$\phi[\hat{y}(\omega)] = \phi[\hat{h}(\omega)] + \phi[\hat{x}(\omega)]$$

```
deltaT = 1; % ms
duration = 1000; % ms
t = 0:deltaT:duration-deltaT;
freqs = [2, 10, 40]; %/1000;
tau = 25;

% Computing impulse and impulse-response function
impulse = zeros(size(t));
impulse(1) = 1;
d = iir_filter(impulse, t, deltaT, tau);

% Initializing amplitude and phase arrays
amp_y = zeros(length(freqs), length(t));
phase_y = zeros(length(freqs), length(t));
sub_plot_inds = 1;

figure();
for ll = 1:length(freqs)
    f = freqs(ll)/1000;
    x = sin(2*pi*f*t); % Computing sinusoid input
    y1 = iir_filter(x, t, deltaT, tau); % Output of sinusoid input

    subplot(3, 1, ll)
    plot(t, y1, 'DisplayName', 'Output');
    hold on;
    plot(t, x, 'DisplayName', 'Input');
    xlabel('Time (ms)')
    ylabel('Amplitude')
    legend('Location', 'northeastoutside')
    x_fft = fft(x)'; % fft of input
    d_fft = fft(d); % fft of impulse-response
    h_fft = 1./(1 + tau .* d_fft); % effective frequency response of IIR

    y_fft = h_fft .* x_fft; % Output sinusoid
    y_fft_shift = fftshift(y_fft);
    amp_y(ll, :) = abs(y_fft_shift); %/tau^2; % Computed amplitude
    phase_y(ll, :) = angle(y_fft_shift); % Computed phase
end

freqHz = (0:1:length(amp_y)/2)/1000;
figure();
for ll = 1:length(freqs)
    subplot(3, 1, ll)
    plot(freqHz, amp_y(ll, length(amp_y)/2:end)) % Plotting amplitude of output for different frequencies
    xlabel('Frequency')
    ylabel('Amplitude')
    title('Amplitude of output for sinusoid')
end

figure();
for ll = 1:length(freqs)
    subplot(3, 1, ll)
    plot(freqHz, phase_y(ll, length(amp_y)/2:end)) % Plotting phase of output for different frequencies
    xlabel('Frequency')
    ylabel('Phase')
    title('Phase of output for sinusoid')
end
```


Amplitude
Time (ms)
Output
Input
Amplitude
Time (ms)
Output
Input
Amplitude
Time (ms)
Output
Input


Amplitude of output for sinusoid
Amplitude
Frequency
Amplitude of output for sinusoid
Amplitude
Frequency
Amplitude of output for sinusoid
Amplitude
Frequency



The output of the IIR filter for the sinusoid is also a sinusoid of the same frequency. However, the amplitude of the resulting sinusoid is scaled down. Additionally, the output sinusoid is also shifted by a certain phase.

```
toc
```

```
Elapsed time is 4.654707 seconds.
```

Published with MATLAB® R2021b

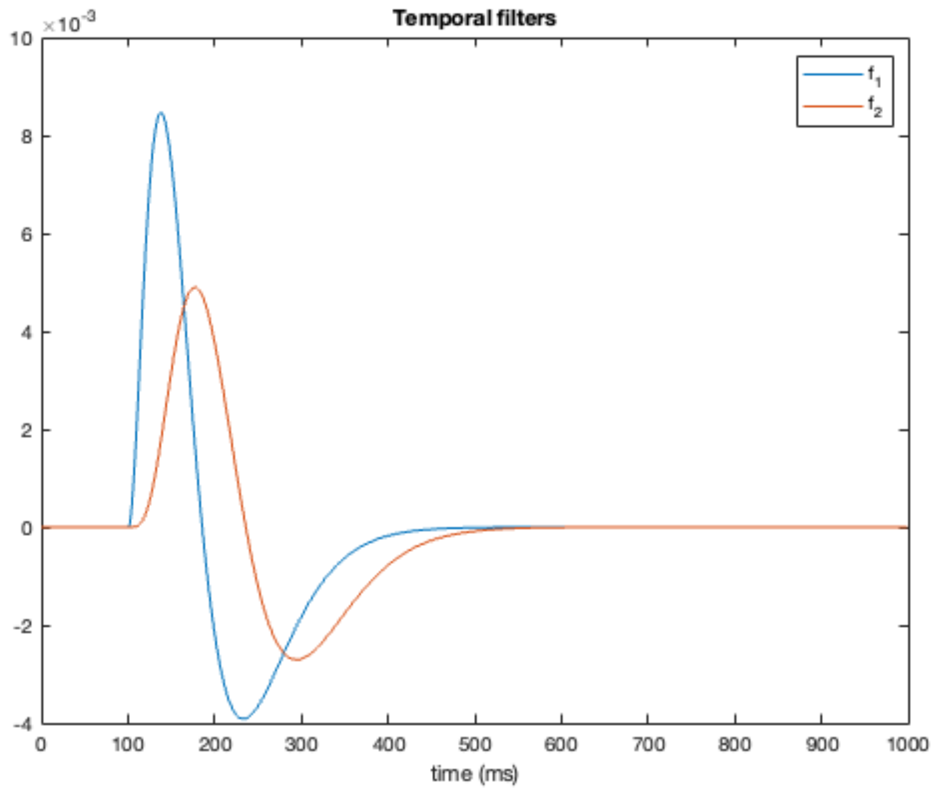
Contents

```
clear; close all; clc;
```

2)

lp_filter uses a cascade of exponential low-pass filters and computes the temporal filters f1 and f2

```
deltaT = 1; % ms
duration = 1000; % ms
t = 0:deltaT:duration-deltaT;
x = zeros(size(t));
x(100) = 1;
tau = 25; % ms
[f1, f2] = lp_filter(x, t, deltaT, tau);

figure();
plot(t, f1, 'DisplayName', 'f_1')
hold on;
plot(t, f2, 'DisplayName', 'f_2')
xlabel('time (ms)')
title('Temporal filters')
legend()
```



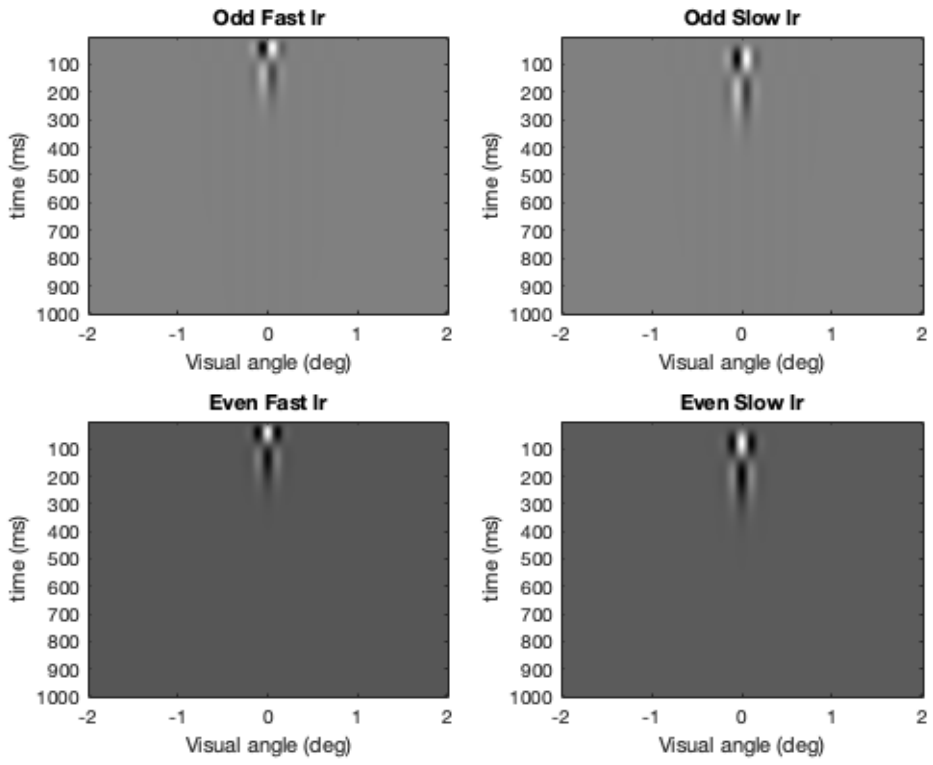
Contents

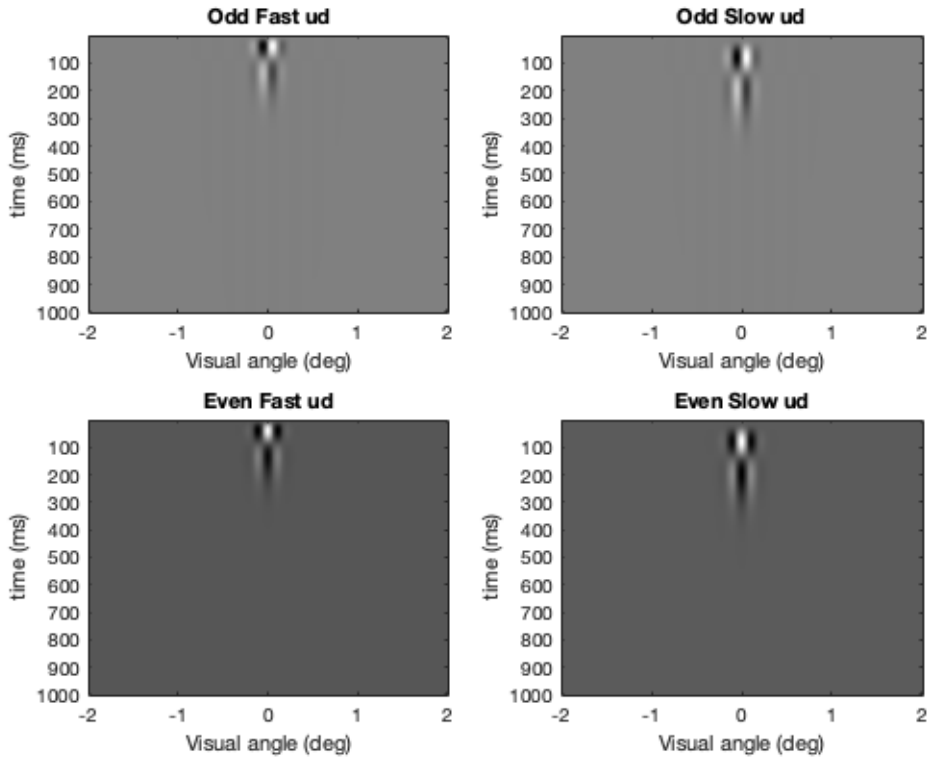
```
clear; close all; clc;
```

a)

Temporal and spatial filters are computed. These filters are then convolved to compute oddFast, oddSlow, evenFast, evenSlow. The orientation of the spatial filters determines the orientation of the convolved outputs. The y-t slices are plotted for horizontal spatial filters and x-t slices are plotted for vertical spatial filters.

```
tic
deltaX = 1/120; % spatial sampling rate
x_x = -2:deltaX:2; % spatial array along x axis
x_y = -2:deltaX:2; % spatial array along y axis
deltaT = 1; % ms
duration = 1000; % ms

t = 0:deltaT:duration-deltaT; % time-array

x = zeros(length(x_x), length(x_y), length(t));
x(241, 241, 1) = 1; % Impulse at center of screen at time = 0

tau = 25; % ms

[f1, f2] = time_filters(x, t, deltaT, tau); % computes temporal filters

sig = 0.1; % standard deviation of the Gaussian (in deg)
sf = 4; % spatial frequency of the sinusoid (in cyc/deg)

[evenFilt, oddFilt] = gabor_filter(x_x, sig, sf); % computes spatial filters

[oddFastlr, oddSlowlr, evenFastlr, evenSlowlr] = ...
    conv_filts(f1, f2, oddFilt, evenFilt); % convolution of temporal and horizontal spatial filters

figure();
plot_func(oddFastlr, oddSlowlr, evenFastlr, evenSlowlr, 'Odd Fast lr', 'Odd Slow lr', ...
    'Even Fast lr', 'Even Slow lr', 'lr')

[oddFastud, oddSlowud, evenFastud, evenSlowud] = ...
    conv_filts(f1, f2, oddFilt', evenFilt'); % convolution of temporal and vertical spatial filters
figure();
plot_func(oddFastud, oddSlowud, evenFastud, evenSlowud, 'Odd Fast ud', 'Odd Slow ud', ...
    'Even Fast ud', 'Even Slow ud', 'ud')
```

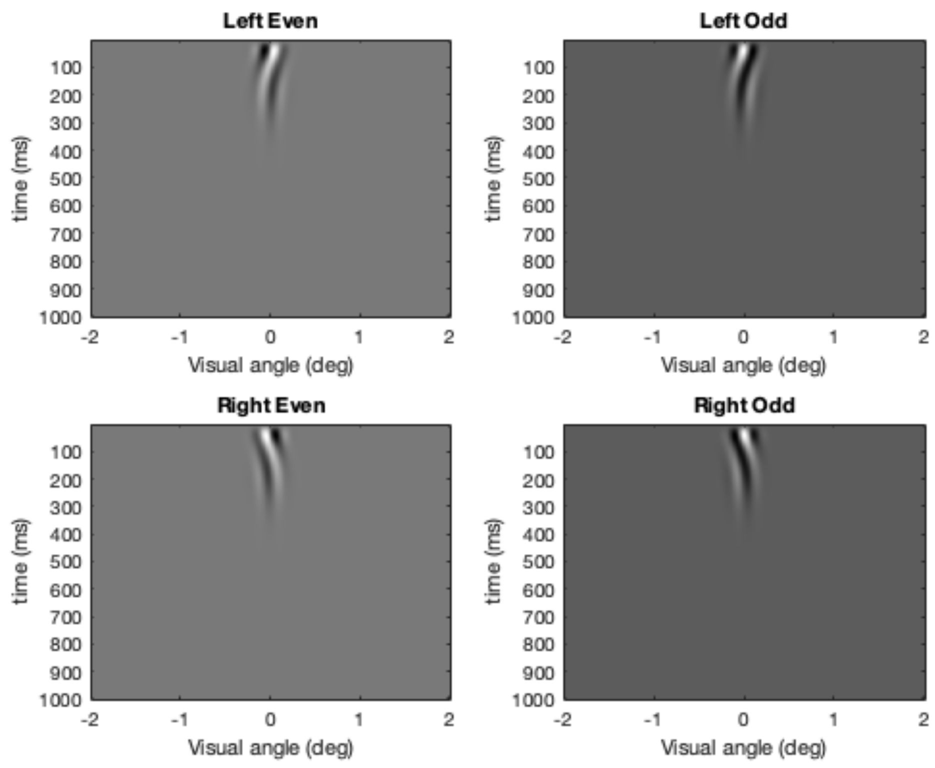


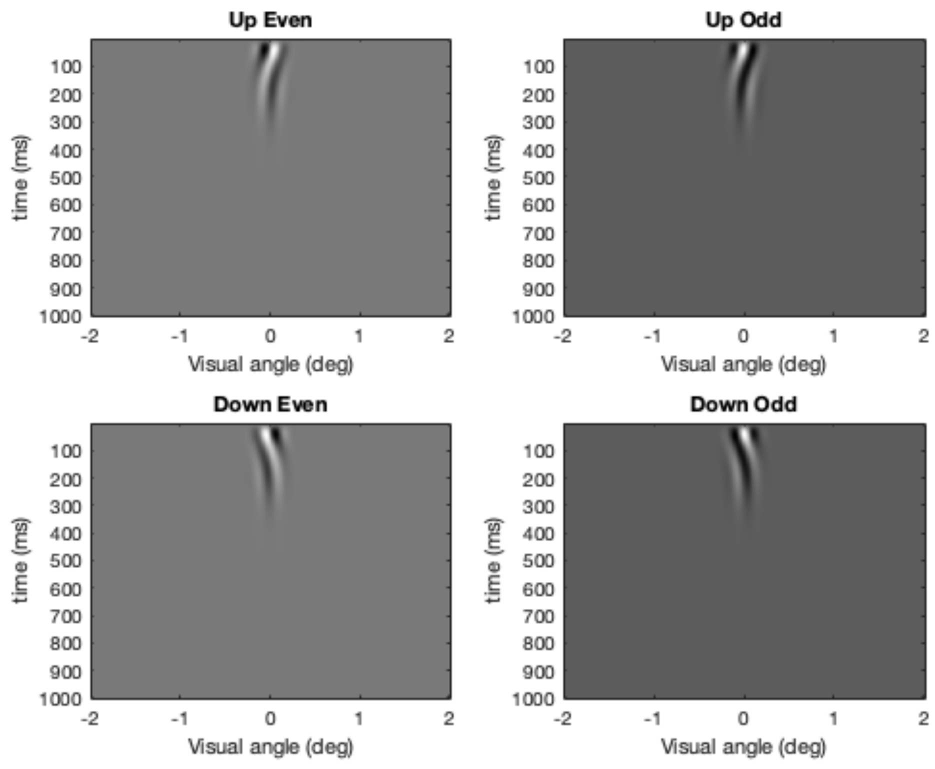


b)

The metrics computed in (a) are then used to compute Even1, Odd1, Even2, Odd2, Energy1, and Energy2 using motion_energy function, where 1 refers to left or up and 2 refers to right or down determined by the orientation of the spatial filters. As expected, the resulting sums obtained appear to curve in time.

```
[leftEven, leftOdd, rightEven, rightOdd, leftEnergy, rightEnergy] = ...
    motion_energy(oddFastlr, oddSlowlr, evenFastlr, evenSlowlr);

[upEven, upOdd, downEven, downOdd, upEnergy, downEnergy] = ...
    motion_energy(oddFastud, oddSlowud, evenFastud, evenSlowud);

figure();
plot_func(leftEven, leftOdd, rightEven, rightOdd, 'Left Even', 'Left Odd', ...
    'Right Even', 'Right Odd', 'lr')

figure();
plot_func(upEven, upOdd, downEven, downOdd, 'Up Even', 'Up Odd', ...
     'Down Even', 'Down Odd', 'ud')
```

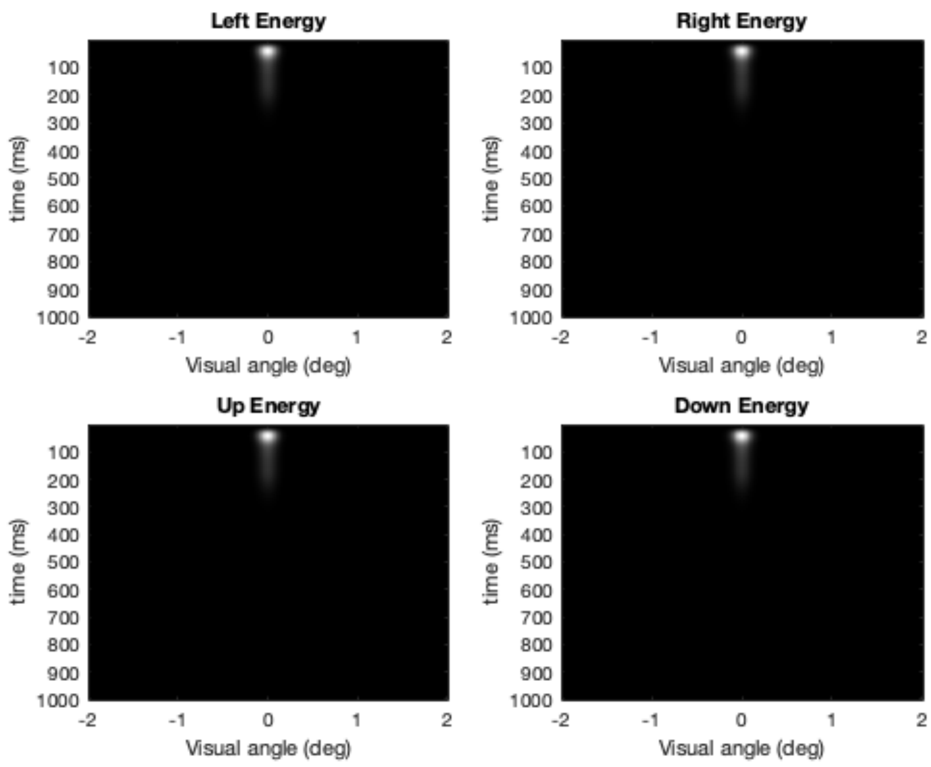




c)

The energies computed in (b) are plotted here.

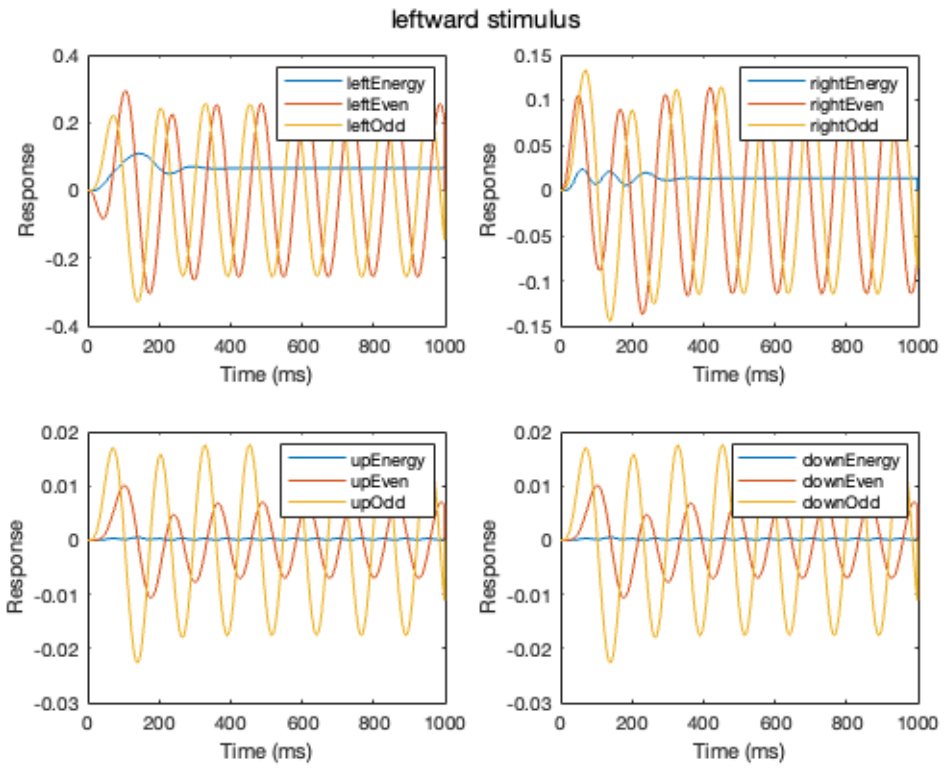
```
figure();
plot_func(leftEnergy, rightEnergy, upEnergy, downEnergy, 'Left Energy', ...
    'Right Energy', 'Up Energy', 'Down Energy', 'energy')
```

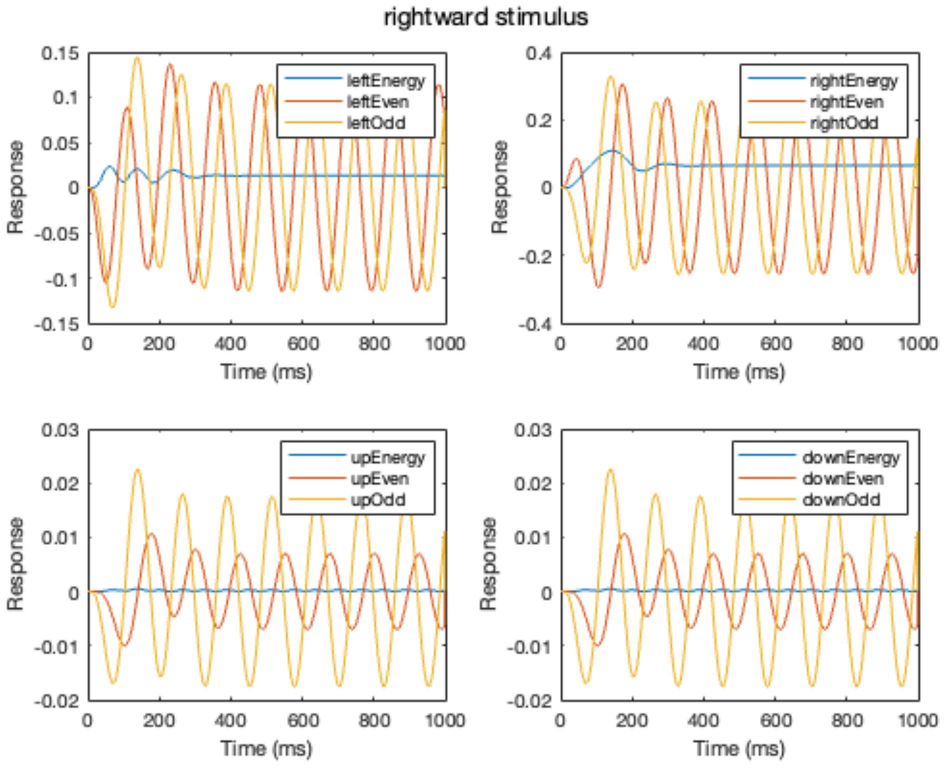


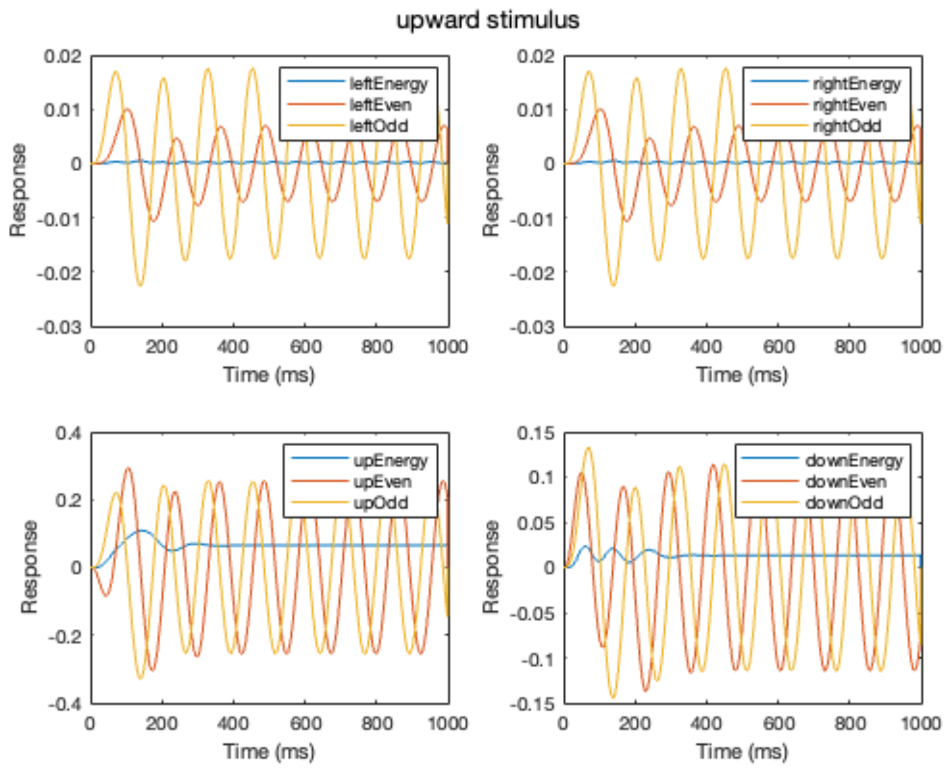
d)

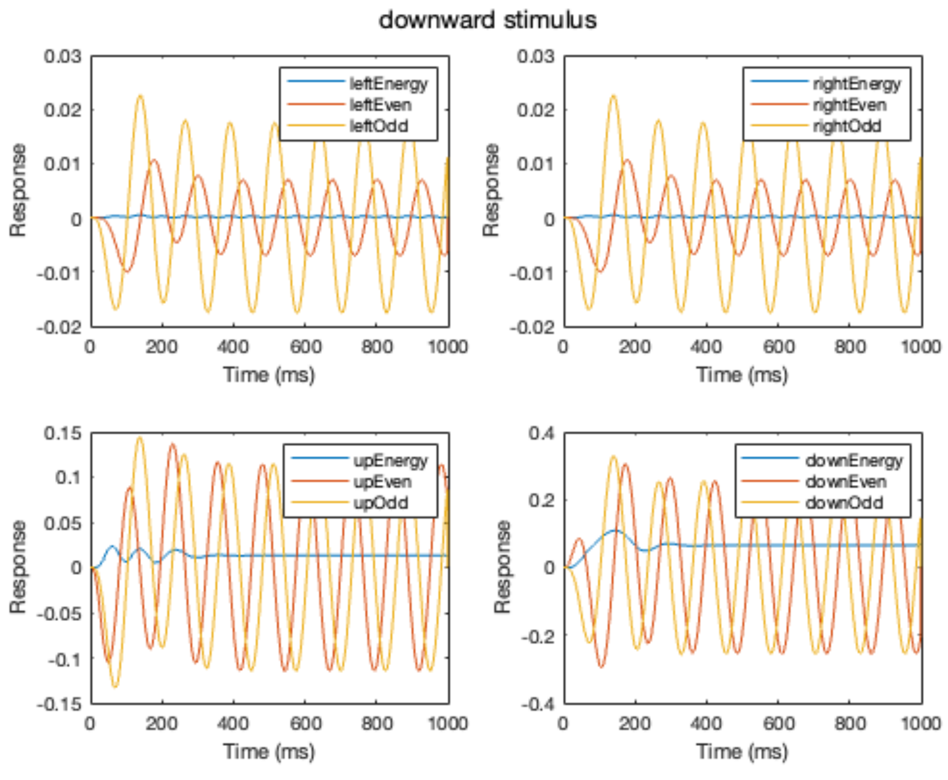
The neuron_responses runs the entire experiment by computing the responses of leftward, rightward, upward, downward oriented neurons for a given series of gratings over time. The gratings are designed using get_grating function that creates a sinusoidal grating in space with the spatial frequency of 4 cyc/deg. Since the spatial scale of the space varies from -2 deg to 2 deg in steps of 1/120 degree. Hence, 4 degrees correspond to 481 pixels. Therefore, the spatial frequency of the sinusoid is 4 cyc/ 481 pixels. Therefore, the time-period of the sinusoid becomes 481/4 ~ 30 pixels/cyc. The sinusoidal grating also moves in time with a frequency of 8Hz = 8 cycles/sec. 1 sec = 1000 ms = 1000 frames. Therefore, in each frame, the sinusoid moves by 1000/8 = 125 frames/cycle. Hence the phase of the sinusoid will change at each time-step by 2*pi/125 in the preferred direction. The sinusoid is then used as an input to compute time_filters which are convolved with the spatial filters to produce neuronal responses and energies.

```
close all; %clearvars -except oddFilt evenFilt deltaX x_x x_y ;

contrast = 100; % contrast of the stimulus
phase = 0; % initial phase of the stimulus
sf = 30; % cycles/pixel

oris = ["lr", "ud"]; % for left-right, and up-down
phase_shifts = [2*pi/125, -2*pi/125]; % cycles/frame, positive is left or down, negative is right or up

for ori = oris
    for phase_shift = phase_shifts
        [leftEven, leftOdd, rightEven, rightOdd, leftEnergy, rightEnergy, ...
            upEven, upOdd, downEven, downOdd, upEnergy, downEnergy] = ...
            neuron_responses(x_x, x_y, t, deltaT, tau, contrast, phase, ...
            phase_shift, sf, ori, oddFilt, evenFilt);

        figure();
        if ori == "lr"
            if phase_shift > 0
                title_st = 'leftward stimulus';
            else
                title_st = 'rightward stimulus';
            end
        else
            if phase_shift > 0
                title_st = 'upward stimulus';
            else
                title_st = 'downward stimulus';
            end
        end
        plot_energy(t, leftEnergy, leftEven, leftOdd, rightEnergy, rightEven, ...
            rightOdd, upEnergy, upEven, upOdd, downEnergy, downEven, downOdd, title_st)
    end
end
```


leftward stimulus
leftEnergy
leftEven
leftOdd
rightEnergy
rightEven
rightOdd
upEnergy
upEven
upOdd
downEnergy
downEven
downOdd
Response
Time (ms)


rightward stimulus
leftEnergy
leftEven
leftOdd
rightEnergy
rightEven
rightOdd
upEnergy
upEven
upOdd
downEnergy
downEven
downOdd
Response
Time (ms)





As can be seen, the neurons respond highest when the grating is moving in their preferred direction.

```
toc
```

```
Elapsed time is 338.928858 seconds.
```

Contents

```
clear; close all; clc;
```

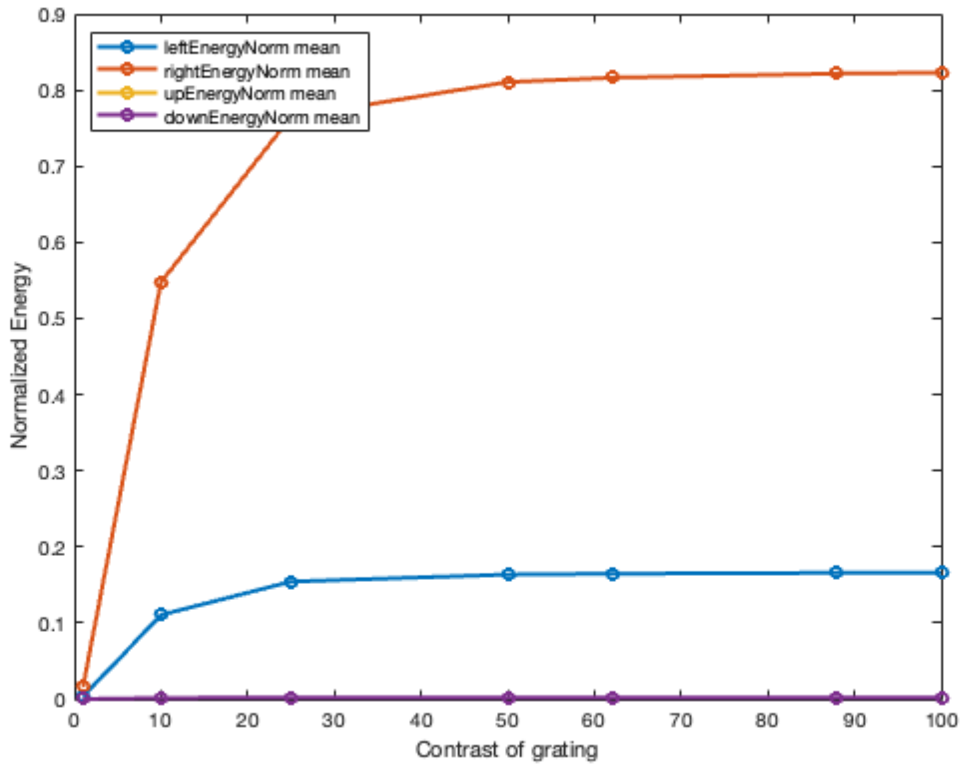
4a)

Here, the sinusoidal grating is moving rightward and the contrast of this grating is varied. For each contrast, the responses of the four direction-selective neurons are computed. The energies are then normalized and this is used to compute the mean energy response for each neuron. The result is a sigmoidal curve. We can see that for the rightward-moving grating, the rightward-selective neuron has the highest energy followed by leftward-selective neuron, while upward and downward selective neurons have almost no response. Thus, we can confirm that the neurons are direction-selective. By changing the contrast, we see that the response of the neurons increases and then saturates to a certain value. The saturation can be controlled by normalizing the energies.

```
tic
deltaX = 1/120; % spatial sampling rate
x_x = -2:deltaX:2; % spatial array along x axis
x_y = -2:deltaX:2; % spatial array along y axis
deltaT = 1; % ms
duration = 1000; % ms

t = 0:deltaT:duration-deltaT; % time-array

tau = 25; % ms

phase = 0; % initial phase of the stimulus
sf = 30; % cycles/pixel

ori = "lr"; % for left-right
phase_shift = -2*pi/125; % cycles/frame
contrasts = [1, 10, 25, 50, 62, 88, 100]; % contrasts of the grating

sig = 0.1; % standard deviation of the Gaussian (in deg)

[evenFilt, oddFilt] = gabor_filter(x_x, sig, 4); % computes spatial filters

mean_energies = zeros(length(contrasts), 4);

for ii = 1:length(contrasts)
    contrast = contrasts(ii);
    [leftEven, leftOdd, rightEven, rightOdd, leftEnergy, rightEnergy, ...
                upEven, upOdd, downEven, downOdd, upEnergy, downEnergy] = ...
                neuron_responses(x_x, x_y, t, deltaT, tau, contrast, phase, ...
                phase_shift, sf, ori, oddFilt, evenFilt);
    [leftEnergyNorm, rightEnergyNorm, upEnergyNorm, downEnergyNorm] = ...
        energy_norm(leftEnergy, rightEnergy, upEnergy, downEnergy);

    x_y_dim = 241;
    st_tm = 400; % ms
    mean_energies(ii, :) = [mean(leftEnergyNorm(x_y_dim, x_y_dim, st_tm:end)), ...
        mean(rightEnergyNorm(x_y_dim, x_y_dim, st_tm:end)), ...
        mean(upEnergyNorm(x_y_dim, x_y_dim, st_tm:end)), ...
        mean(downEnergyNorm(x_y_dim, x_y_dim, st_tm:end))];
end
toc
```

```
plt_titles = ["leftEnergyNorm mean", "rightEnergyNorm mean", ...
    "upEnergyNorm mean", "downEnergyNorm mean"];

figure()
for ss = 1:size(mean_energies, 2)
    plot(contrasts, mean_energies(:, ss), 'o-', ...
        'DisplayName', plt_titles(ss), 'LineWidth', 2);
    xlabel('Contrast of grating')
    ylabel('Normalized Energy')
    legend('location', 'northwest')
    hold on;
end
```

```
Elapsed time is 478.336377 seconds.
```



Published with MATLAB® R2021b

Contents

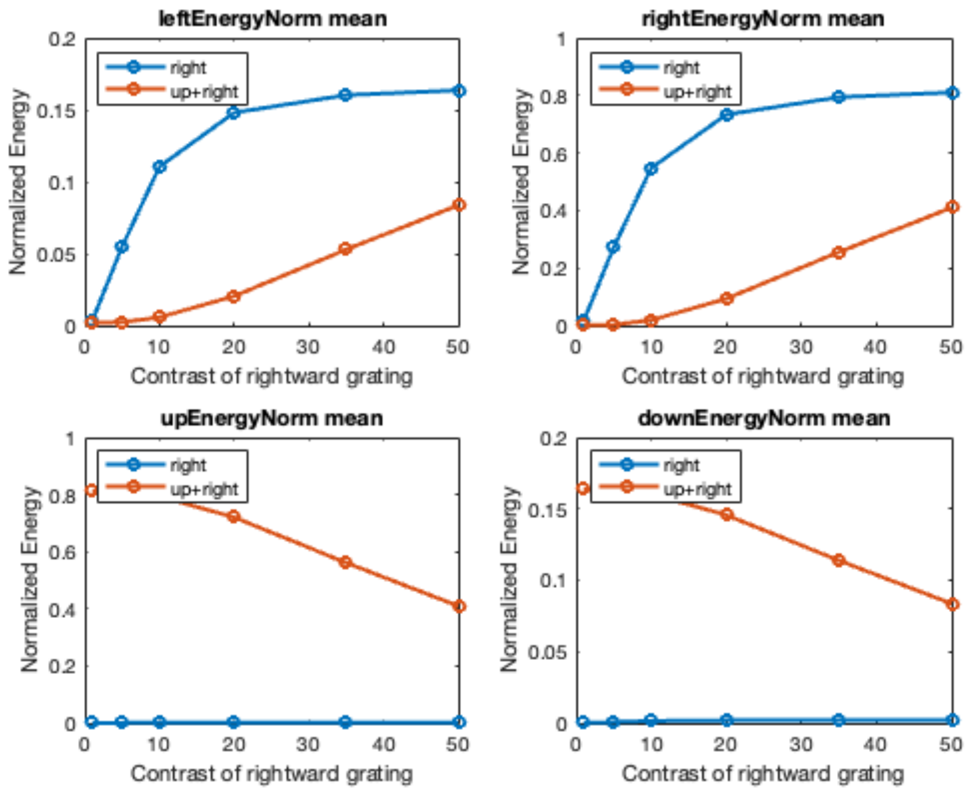
```
clear; close all; clc;
```

b)

Repeating the experiment for a cross-orientation experiment where the contrast of the rightward moving grating is varied between 1% and 50% and the contrast of the upward moving grating is kept at 50%. The normalized energies of each of the direction-selective neurons are computed for different contrasts of the rightward moving gratins when the upward moving grating is either present or absent. The rightward-selective neuron responds the strongest when there is only rightward moving grating and the response is dampened by the presence of the upward selective grating. However, the effect of upward grating decreases with an increase in the contrast of the rightward moving grating. A similar pattern is observed for the leftward selective response. However, the scale of the responses for the leftward selective neuron are lower than the rightward selective neuron as the stimulus is not along the preferred direction of the neuron. For the upward selective neuron, the response is higher when the contrast of the rightward moving grating is lower and the responses decrease when the contrast of the rightward moving grating increases. Similar pattern is observed for the downward selective neuron but the responses are lower than the upward selective neuron as the stimulus is not along the preferred direction of the neuron.

```
tic
deltaX = 1/120; % spatial sampling rate
x_x = -2:deltaX:2; % spatial array along x axis
x_y = -2:deltaX:2; % spatial array along y axis
deltaT = 1; % ms
duration = 1000; % ms

t = 0:deltaT:duration-deltaT; % time-array

tau = 25; % ms

phase_shift = 2*pi/125;
contrasts = [1, 5, 10, 20, 35, 50];
cross_oris = [0, 1];
phase = 0; % initial phase of the stimulus
sf = 30; % cycles/pixel
sig = 0.1; % standard deviation of the Gaussian (in deg)

[evenFilt, oddFilt] = gabor_filter(x_x, sig, 4); % computes spatial filters

ori = "lr"; % for left-right

mean_energies = zeros(length(contrasts), 4, length(cross_oris));

for ff = 1:2
    cross_ori = cross_oris(ff);
    for ii = 1:length(contrasts)
        contrast = contrasts(ii);
        [leftEven, leftOdd, rightEven, rightOdd, leftEnergy, rightEnergy, ...
                upEven, upOdd, downEven, downOdd, upEnergy, downEnergy] = ...
                neuron_responses_cross_ori(x_x, x_y, t, deltaT, tau, contrast, phase, ...
                phase_shift, sf, oddFilt, evenFilt, cross_ori);
        [leftEnergyNorm, rightEnergyNorm, upEnergyNorm, downEnergyNorm] = ...
            energy_norm(leftEnergy, rightEnergy, upEnergy, downEnergy);
        x_y_dim = 241;
        st_tm = 400; % ms
        mean_energies(ii, :, ff) = [mean(leftEnergyNorm(x_y_dim, x_y_dim, st_tm:end)), ...
            mean(rightEnergyNorm(x_y_dim, x_y_dim, st_tm:end)), ...
            mean(upEnergyNorm(x_y_dim, x_y_dim, st_tm:end)), ...
            mean(downEnergyNorm(x_y_dim, x_y_dim, st_tm:end))];
    end
end
toc

plt_titles = ["leftEnergyNorm mean", "rightEnergyNorm mean", ...
    "upEnergyNorm mean", "downEnergyNorm mean"];

figure()
for ss = 1:size(mean_energies, 2)
    subplot(2, 2, ss)
    plot(contrasts, mean_energies(:, ss, 1), 'o-', ...
        'DisplayName', 'right', 'LineWidth', 2);
    legend('location', 'northwest')
    hold on;
    plot(contrasts, mean_energies(:, ss, 2), 'o-', ...
        'DisplayName', 'up+right', 'LineWidth', 2);
    xlabel('Contrast of rightward grating')
    ylabel('Normalized Energy')
    title(plt_titles(ss))
end
```

```
Elapsed time is 898.022641 seconds.
```



iir_filter

Given the input x, time_array t, time-step deltaT, and time-constant tau, the function solves the ODE to compute the response obtained throught the IIR-filter

```
function y1 = iir_filter(x, t, deltaT, tau)
    y1 = zeros(length(t), 1);
    for tt = 1:length(t) - 1
        deltaY1 = (deltaT/tau) * (-y1(tt) + x(tt));
        y1(tt + 1) = y1(tt) + deltaY1;
    end
end
```

Published with MATLAB® R2021b

low-pass filter

Given the input x, time_array t, time-step deltaT, and time-constant tau, the function solves a cascade of ODEs to compute the temporal filters f1 and f2. Here f1 is a fast-filter while f2 is a slow filter.

```
function [f1, f2] = lp_filter(x, t, deltaT, tau)

    y = zeros(length(t), 7);

    for tt = 1:length(t) - 1
        for jj = 1:7
            if jj == 1
                deltaY = (deltaT / tau) * (-y(tt, jj) + x(tt));
                y(tt + 1, jj) = y(tt, jj) + deltaY;
            else
                deltaY = (deltaT / tau) * (-y(tt, jj) + y(tt, jj - 1));
                y(tt + 1, jj) = y(tt, jj) + deltaY;
            end
        end
    end
    f1 = y(:, 3) - y(:, 5); % Fast filter
    f2 = y(:, 5) - y(:, 7); % Slow filter
end
```

Published with MATLAB® R2021b

gabor filter

Given a spatial resolution x, the standard deviation of the gaussian sigma, and the spatial frequency sf, gabor_filter computes even and odd Gabor filters by multiplying a Gaussian with a cosine and a sine, respectively. It then normalizes it using the sum of squares of evenFilt and oddFilt.

```
function [evenFilt, oddFilt] = gabor_filter(x, sig, sf)
    evenFilt = exp(-(x.^2)./(2*sig^2)) .* cos(2*pi*sf*x);
    oddFilt = exp(-(x.^2)./(2*sig^2)) .* sin(2*pi*sf*x);
    integral = sum(evenFilt.^2 + oddFilt.^2);
    evenFilt = evenFilt / integral;
    oddFilt = oddFilt / integral;
end
```

Published with MATLAB® R2021b

convolve filters

The temporal filters f1 (fast) and f2 (slow) are convolved with the spatial filters oddFilt and evenFilt to produce oddFast, oddSlow, evenFast, and evenSlow at each time-point. This algorithm uses conv2 to compute 2-dimensional convolutions at each time-point.

```
function [oddFast, oddSlow, evenFast, evenSlow] = ...
    conv_filts(f1, f2, oddFilt, evenFilt)
    [lx, ly, lt] = size(f1);
    oddFast = zeros(lx, ly, lt);
    oddSlow = zeros(lx, ly, lt);
    evenSlow = zeros(lx, ly, lt);
    evenFast = zeros(lx, ly, lt);

    for tt = 1:lt
        oddFast(:, :, tt) = conv2(f1(:, :, tt), oddFilt, 'same');
        oddSlow(:, :, tt) = conv2(f2(:, :, tt), oddFilt, 'same');
        evenSlow(:, :, tt) = conv2(f2(:, :, tt), evenFilt, 'same');
        evenFast(:, :, tt) = conv2(f1(:, :, tt), evenFilt, 'same');
    end
end
```

Published with MATLAB® R2021b

convolve filters by convn

The temporal filters f1 (fast) and f2 (slow) are convolved with the spatial filters oddFilt and evenFilt to produce oddFast, oddSlow, evenFast, and evenSlow at each time-point. This algorithm uses convn to compute 3-dimensional convolutions. The resulting array is 3-dimensional with the third-axis being time.

```matlab
function [oddFast, oddSlow, evenFast, evenSlow] = ...
    conv_filts_by_convn(f1, f2, oddFilt, evenFilt)
    oddFast = convn(f1, oddFilt, 'same');
    oddSlow = convn(f2, oddFilt, 'same');
    evenSlow = convn(f2, evenFilt, 'same');
    evenFast = convn(f1, evenFilt, 'same');
end
```

Published with MATLAB® R2021b

convolve filters by fft

The temporal filters f1 (fast) and f2 (slow) are convolved with the spatial filters oddFilt and evenFilt to produce oddFast, oddSlow, evenFast, and evenSlow at each time-point. This algorithm uses fft2 to compute 2-dimensional fft's for each filter and then computes ifft2 of the element-wise dot-product of the fft's of spatial and temporal filters at each time-point.

```
function [oddFast, oddSlow, evenFast, evenSlow] = ...
    conv_filts_by_fft(f1, f2, oddFilt, evenFilt)
    [lx, ly, lt] = size(f1);
    oddFast = zeros(lx, ly, lt);
    oddSlow = zeros(lx, ly, lt);
    evenSlow = zeros(lx, ly, lt);
    evenFast = zeros(lx, ly, lt);

    for tt = 1:lt
        oddFast(:, :, tt) = ifft2(fft2(f1(:, :, tt)) .* fft2(oddFilt));
        oddSlow(:, :, tt) = ifft2(fft2(f2(:, :, tt)) .* fft2(oddFilt));
        evenSlow(:, :, tt) = ifft2(fft2(f2(:, :, tt)) .* fft2(evenFilt));
        evenFast(:, :, tt) = ifft2(fft2(f1(:, :, tt)) .* fft2(evenFilt));
    end
end
```

Published with MATLAB® R2021b

convolve filters by fftn

The temporal filters f1 (fast) and f2 (slow) are convolved with the spatial filters oddFilt and evenFilt to produce oddFast, oddSlow, evenFast, and evenSlow at each time-point. This algorithm uses fftn to compute 3-dimensional fft's for each filter and then computes ifftn of the element-wise dot-product of the fft's of spatial and temporal filters. The resulting array is 3-dimensional with the third-axis being time.

```
function [oddFast, oddSlow, evenFast, evenSlow] = ...
    conv_filts_by_fftn(f1, f2, oddFilt, evenFilt)

    oddFast = ifftn(fftn(f1) .* fftn(oddFilt));
    oddSlow = ifftn(fftn(f2) .* fftn(oddFilt));
    evenSlow = ifftn(fftn(f2) .* fftn(evenFilt));
    evenFast = ifftn(fftn(f1) .* fftn(evenFilt));
end
```

Published with MATLAB® R2021b

motion energy

Given oddFast, oddSlow, evenFast, evenSlow, the function computes Even1, Odd1, Even2, Odd2, where 1 refers to left/up and 2 refers to right/down depending on the orientation of the grating and the direction in which it is moving. These Even's and Odd's are then used to compute the corresponding Energies by taking a sum of element-wise square of the elements.

```
function [Even1, Odd1, Even2, Odd2, Energy1, Energy2]...
    = motion_energy(oddFast, oddSlow, evenFast, evenSlow)
    Even1 = oddFast + evenSlow;
    Odd1 = -oddSlow + evenFast;
    Even2 = -oddFast + evenSlow;
    Odd2 = oddSlow + evenFast;
    Energy1 = Even1.^2 + Odd1.^2;
    Energy2 = Even2.^2. + Odd2.^2;
end
```

Published with MATLAB® R2021b

create 2-D sinusoid

The function creates a 2-D sinusoid with given amplitude, phase, spatial frequency (sf), orientation ('lr' for horizontal, and 'ud' for vertical), and the range of x and y for the sinusoid is determined by a one-dimensional array x_x.

```
function sinewave2D = get_grating(x_x, amp, phase, sf, ori)
    [X,Y] = meshgrid(x_x .* sf);
    if strcmp(ori, 'lr')
        sinewave2D = amp * sin(X + phase);
    elseif strcmp(ori, 'ud')
        sinewave2D = amp * sin(Y + phase);
    end
end
```

Published with MATLAB® R2021b

energy normalization

The function computes normalized energies for each neuron by normalizing it with the sum of energies of responses for all the neurons. stdev here controls the normalization.

```
function [leftEnergyNorm, rightEnergyNorm, upEnergyNorm, downEnergyNorm] = ...
    energy_norm(leftEnergy, rightEnergy, upEnergy, downEnergy)
    sumEnergy = leftEnergy + rightEnergy + upEnergy + downEnergy;
    stdev = 0.02;
    leftEnergyNorm = leftEnergy ./ (sumEnergy + stdev^2);
    rightEnergyNorm = rightEnergy ./ (sumEnergy + stdev^2);
    upEnergyNorm = upEnergy ./ (sumEnergy + stdev^2);
    downEnergyNorm = downEnergy ./ (sumEnergy + stdev^2);
end
```

Published with MATLAB® R2021b

neuron response

The neuron_responses runs the entire experiment by computing the responses of leftward, rightward, upward, downward oriented neurons for a given series of gratings over time. The gratings are designed using get_grating function that creates a sinusoidal grating in space with the spatial frequency of 4 cyc/deg. Since the spatial scale of the space varies from -2 deg to 2 deg in steps of 1/120 degree. Hence, 4 degrees correspond to 481 pixels. Therefore, the spatial frequency of the sinusoid is 4 cyc/ 481 pixels. Therefore, the time-period of the sinusoid becomes 481/4 ~ 30 pixels/cyc. The sinusoidal grating also moves in time with a frequency of 8Hz = 8 cycles/sec. 1 sec = 1000 ms = 1000 frames. Therefore, in each frame, the sinusoid moves by 1000/8 = 125 frames/cycle. Hence the phase of the sinusoid will change at each time-step by 2*pi/125 in the preferred direction. The sinusoid is then used as an input to compute time_filters which are convolved with the spatial filters to produce neuronal responses and energies.

```
function [leftEven, leftOdd, rightEven, rightOdd, leftEnergy, rightEnergy, ...
            upEven, upOdd, downEven, downOdd, upEnergy, downEnergy] = ...
            neuron_responses(x_x, x_y, t, deltaT, tau, contrast, phase, ...
            phase_shift, sf, ori, oddFilt, evenFilt)

    sinusoid_x = zeros(length(x_x), length(x_y), length(t)); % Initializing the sinusoid

    amp_sinusoid = contrast * 1/100; % amplitude of the sinusoid is scaled by the contrast of the stimulus desired

    for tt = 1:1000
        phase = phase + phase_shift; % Shifting the phase at each time-step to move the gratings
        sinusoid_x(:, :, tt) = get_grating(x_x, amp_sinusoid, phase, sf, ori); % Generating 2-D sinusoid at each time-step
    end

    [f1, f2] = time_filters(sinusoid_x, t, deltaT, tau); % Obtain temporal filters with sinusoid as input

    [oddFastlr, oddSlowlr, evenFastlr, evenSlowlr] = ...
        conv_filts(f1, f2, oddFilt, evenFilt); % convolve temporal and horizontal spatial filters
    [oddFastud, oddSlowud, evenFastud, evenSlowud] = ...
        conv_filts(f1, f2, oddFilt', evenFilt'); % convolve temporal and vertical spatial filters

    [leftEven, leftOdd, rightEven, rightOdd, leftEnergy, rightEnergy] = ...
        motion_energy(oddFastlr, oddSlowlr, evenFastlr, evenSlowlr); % compute even, odd, energy for leftward and rightward oriented neurons
    [upEven, upOdd, downEven, downOdd, upEnergy, downEnergy] = ...
        motion_energy(oddFastud, oddSlowud, evenFastud, evenSlowud); % compute even, odd, energy for upward and downward oriented neurons
end
```

Published with MATLAB® R2021b

neuron response for cross-orientation

Repeating the experiment for a cross-orientation experiment where the contrast of the rightward moving grating is varied between 1% and 50% and the contrast of the upward moving grating is kept at 50%. The normalized energies of each of the direction-selective neurons are computed for different contrasts of the rightward moving gratins when the upward moving grating is either present or absent. The rightward-selective neuron responds the strongest when there is only rightward moving grating and the response is dampened by the presence of the upward selective grating. However, the effect of upward grating decreases with an increase in the contrast of the rightward moving grating. A similar pattern is observed for the leftward selective response. However, the scale of the responses for the leftward selective neuron are lower than the rightward selective neuron as the stimulus is not along the preferred direction of the neuron. For the upward selective neuron, the response is higher when the contrast of the rightward moving grating is lower and the responses decrease when the contrast of the rightward moving grating increases. Similar pattern is observed for the downward selective neuron but the responses are lower than the upward selective neuron as the stimulus is not along the preferred direction of the neuron.

```
function [leftEven, leftOdd, rightEven, rightOdd, leftEnergy, rightEnergy, ...
            upEven, upOdd, downEven, downOdd, upEnergy, downEnergy] = ...
            neuron_responses_cross_ori(x_x, x_y, t, deltaT, tau, contrast, phase, ...
            phase_shift, sf, oddFilt, evenFilt, cross_ori)

    stim = zeros(length(x_x), length(x_y), length(t)); % Initializing the stimulus

    amp_sinusoid_right = contrast * 1/100; % amplitude of the rightward moving sinusoid is scaled by the contrast of the stimulus desired
    amp_sinusoid_up = 0.5; % amplitude of the upward moving sinusoid is 50%
    phase_up = phase; % Initializing phase of upward moving sinusoid
    phase_right = phase; % Initializing phase of rightward moving sinusoid
    if cross_ori == 1

        for tt = 1:length(t)
            phase_right = phase_right - phase_shift; % Shifting the phase at each time-step to move the gratings rightward
            phase_up = phase_up + phase_shift; % Shifting the phase at each time-step to move the gratings upward
            rightwardsinusoid = get_grating(x_x, amp_sinusoid_right, phase_right,...
                sf, "lr"); % Generating rightward moving 2-D sinusoid at each time-step
            upwardsinusoid = get_grating(x_x, amp_sinusoid_up, phase_up,...
                sf, "ud"); % Generating upward moving 2-D sinusoid at each time-step
            stim(:, :, tt) = rightwardsinusoid + upwardsinusoid; % Stimulus is the sum of the two gratings
        end

    else
        for tt = 1:length(t)
            phase_right = phase_right - phase_shift; % Shifting the phase at each time-step to move the gratings rightward
            rightwardsinusoid = get_grating(x_x, amp_sinusoid_right, phase_right,...
                sf, "lr"); % Generating rightward moving 2-D sinusoid at each time-step
            stim(:, :, tt) = rightwardsinusoid; % Stimulus is the rightward moving grating
        end

    end
    [f1, f2] = time_filters(stim, t, deltaT, tau); % Obtain temporal filters with sinusoid as input

    [oddFastlr, oddSlowlr, evenFastlr, evenSlowlr] = ...
        conv_filts(f1, f2, oddFilt, evenFilt); % convolve temporal and horizontal spatial filters
    [oddFastud, oddSlowud, evenFastud, evenSlowud] = ...
        conv_filts(f1, f2, oddFilt', evenFilt'); % convolve temporal and vertical spatial filters

    [leftEven, leftOdd, rightEven, rightOdd, leftEnergy, rightEnergy] = ...
        motion_energy(oddFastlr, oddSlowlr, evenFastlr, evenSlowlr); % compute even, odd, energy for leftward and rightward oriented neurons
    [upEven, upOdd, downEven, downOdd, upEnergy, downEnergy] = ...
        motion_energy(oddFastud, oddSlowud, evenFastud, evenSlowud); % compute even, odd, energy for upward and downward oriented neurons

end
```

plotting

The function creates 4-subplots: 1 for each input passed. Based on the orientation of the temporal filter used in computing the inputs, it will create a x-t/y-t slice for visualization. If, however, the inputs are energies, it will plot the left and right energies by taking x-t slices and the up and down energies by taking y-t slices.

```
function plot_func(a, b, c, d, a_title, b_title, c_title, d_title, direc)

    if strcmp(direc, 'lr') % for left-right
        a = squeeze(a(241, :, :))';
        b = squeeze(b(241, :, :))';
        c = squeeze(c(241, :, :))';
        d = squeeze(d(241, :, :))';
    elseif strcmp(direc, 'ud') % for up-down
        a = squeeze(a(:, 241, :))';
        b = squeeze(b(:, 241, :))';
        c = squeeze(c(:, 241, :))';
        d = squeeze(d(:, 241, :))';
    elseif strcmp(direc, 'energy')
        a = squeeze(a(241, :, :))';
        b = squeeze(b(241, :, :))';
        c = squeeze(c(:, 241, :))';
        d = squeeze(d(:, 241, :))';
    end

    subplot(2,2,1)
    imagesc(a)
    colormap(gray)
    xticks([1, 121, 241, 361, 481])
    xticklabels([-2, -1, 0, 1, 2])
    yticks(0:100:1000)
    xlabel('Visual angle (deg)')
    ylabel('time (ms)')
    title(a_title)

    subplot(2,2,2)
    imagesc(b)
    colormap(gray)
    xticks([1, 121, 241, 361, 481])
    xticklabels([-2, -1, 0, 1, 2])
    yticks(0:100:1000)
    xlabel('Visual angle (deg)')
    ylabel('time (ms)')
    title(b_title)

    subplot(2,2,3)
    imagesc(c)
    colormap(gray)
    xticks([1, 121, 241, 361, 481])
    xticklabels([-2, -1, 0, 1, 2])
    yticks(0:100:1000)
    xlabel('Visual angle (deg)')
    ylabel('time (ms)')
    title(c_title)

    subplot(2,2,4)
    imagesc(d)
    colormap(gray)
    xticks([1, 121, 241, 361, 481])
    xticklabels([-2, -1, 0, 1, 2])
    yticks(0:100:1000)
    xlabel('Visual angle (deg)')
    ylabel('time (ms)')
    title(d_title)
end
```

Published with MATLAB® R2021b

plotting energies

The function creates 4-subplots, each corresponding to the preferred direction of the neuron. Each subplot has 3 lines: even, odd and energy. For each line-plot, a time-slice is taken for the center of the screen with (x, y) = (0 deg, 0 deg)

```
function plot_energy(t, leftEnergy, leftEven, leftOdd, rightEnergy, rightEven, ...
    rightOdd, upEnergy, upEven, upOdd, downEnergy, downEven, downOdd, title_st)

    sgtitle(title_st)
    pp = 241; % the x and y co-ords for 0 degree spatial angle

    subplot(2, 2, 1)
    plot(t, squeeze(leftEnergy(pp, pp, :)), 'DisplayName', 'leftEnergy');
    hold on;
    plot(t, squeeze(leftEven(pp, pp, :)), 'DisplayName', 'leftEven');
    plot(t, squeeze(leftOdd(pp, pp, :)), 'DisplayName', 'leftOdd');
    xlabel('Time (ms)')
    ylabel('Response')
    legend()

    subplot(2, 2, 2)
    plot(t, squeeze(rightEnergy(pp, pp, :)), 'DisplayName', 'rightEnergy');
    hold on;
    plot(t, squeeze(rightEven(pp, pp, :)), 'DisplayName', 'rightEven');
    plot(t, squeeze(rightOdd(pp, pp, :)), 'DisplayName', 'rightOdd');
    xlabel('Time (ms)')
    ylabel('Response')
    legend()

    subplot(2, 2, 3)
    plot(t, squeeze(upEnergy(pp, pp, :)), 'DisplayName', 'upEnergy');
    hold on;
    plot(t, squeeze(upEven(pp, pp, :)), 'DisplayName', 'upEven');
    plot(t, squeeze(upOdd(pp, pp, :)), 'DisplayName', 'upOdd');
    xlabel('Time (ms)')
    ylabel('Response')
    legend()

    subplot(2, 2, 4)
    plot(t, squeeze(downEnergy(pp, pp, :)), 'DisplayName', 'downEnergy');
    hold on;
    plot(t, squeeze(downEven(pp, pp, :)), 'DisplayName', 'downEven');
    plot(t, squeeze(downOdd(pp, pp, :)), 'DisplayName', 'downOdd');
    xlabel('Time (ms)')
    ylabel('Response')
    legend()
end
```

Published with MATLAB® R2021b