

Amplitudenbestimmung

Ausgangslage:

$f_{\text{sin}} = 1000 \text{ Hz}$

$f_s = 2300 \text{ Hz}$

Offset = 0

Phase = ϕ , const

Amplitude unbekannt.

Code

```
rng default

% Signal generation
fsin = 1000;
fs = 2300;
numPeriods = 10; % N periods to plot

a = 5+10*rand(1); % amplitude, positive
phi = pi/2*rand(1); % phase, between 0 and pi/2 rad

dt = 1e-6; % for signal plotting only
t = 0:dt:numPeriods/fsin; % time vector for plotting

sig = a .* sin(2*pi*fsin.*t + phi) + 2*randn(size(t));

tStart = rand(1)/1000; % start sampling somewhere between 0 and 0.001 seconds
tSample = tStart:1/fs:t(end); % sample the signal with fs
sigSample = interp1(t, sig, tSample); % extract the signal at given sample
                                        % points. Use interp, because samplepoint
                                        % may be between two signal sampling points

tPeriods = [1:numPeriods].*1/fsin; % [1T 2T 3T ...]
```

Amplitude + Phase estimation

```
% maximum of sample points
x1(1) = max(sigSample);
x1(2) = nan;

% nonlinear optimization, all points
fun = @(x)x(1)*sin(2*pi*fsin.*tSample + x(2)) - sigSample; % objective function for nonlin
                                                            % optimization
```

```
x2 = lsqnonlin(fun,[0 pi/4],[-inf 0],[inf pi]); % restrict phase to be in [0 pi]
```

Local minimum possible.

lsqnonlin stopped because the final change in the sum of squares relative to its initial value is less than the default value of the function tolerance.

<stopping criteria details>

```
% 2 points, two equations:
% y1 = a*sin(2*pi*f*sin*t1 + phi)
% y2 = a*sin(2*pi*f*sin*t2 + phi)
fun2= @(x)...
    [x(1)*sin(2*pi*f*sin*tSample(1) + x(2)) - sigSample(1);
    x(1)*sin(2*pi*f*sin*tSample(2) + x(2)) - sigSample(2)];

x3 = fsolve(fun2,[0 pi/4]); % I'm too lazy to solve analytically for a and phi,
```

Equation solved.

fsolve completed because the vector of function values is near zero as measured by the default value of the function tolerance, and the problem appears regular as measured by the gradient.

<stopping criteria details>

```
% but that should be possible, too.
```

```
% print errors
fprintf("Errors:\n")
```

Errors:

```
fprintf("Maximum of samples")
```

Maximum of samples

```
fprintf("Error in Amplitude: %f\nError in Phase: %f", x1(1)-a, x1(2)-phi)
```

```
Error in Amplitude: 1.044038
Error in Phase: NaN
```

```
fprintf("Nonlinear optimization:")
```

Nonlinear optimization:

```
fprintf("Error in Amplitude: %f\nError in Phase: %f", x2(1)-a, x2(2)-phi)
```

```
Error in Amplitude: 0.595325
Error in Phase: 0.048975
```

```
fprintf("2 Equation system:")
```

2 Equation system:

```
fprintf("Error in Amplitude: %f\nError in Phase: %f", x3(1)-a, x3(2)-phi)
```

Error in Amplitude: 1.044269
Error in Phase: 0.169726

Plot the stuff

```
figure, hold all  
  
plot(t, sig, 'LineWidth', 1.5, 'Color', [.8 .8 .8]);  
plot(tSample, sigSample, 'x', 'LineWidth', 2, 'Color', 'k')  
  
line(repmat(tPeriods, 2, 1), repmat([-20 20]', 1, numPeriods), 'color', [.5 .5 .5], ...  
     'linestyle', '--');  
  
ylim([min(sig)-1,max(sig)+1])  
set(gca, 'YGrid', 'on')  
xlabel('time in sec')  
ylabel('amplitude')  
  
legend('Real sine', 'Sampling points')
```



