# sinoids2angles

Computes angles in rad or degree by passed sinoids and radius. The angle calculation is asigned in two steps. At first compute bare angles by acos and asin functions. Divide therfore the corresponding sinoid by the radius. Furhter on the result from acos is used with an interval correction given by the second results from the asin function. The amplitudes of the sinoids must be one or near to one.

So angle computation relates to the unit circle with depencies of

$$f_{rad} = \sqrt{f_{sin}^2 + f_{cos}^2}$$

and computes intermediate angle results in rad of

$$f_c = \arccos\left(\frac{f_{cos}}{f_{rad}}\right)$$

$$f_s = \arcsin\left(\frac{f_{sin}}{f_{rad}}\right)$$

$$f_a = \arctan 2\left(\frac{f_{sin}}{f_{cos}}\right)$$

The final angle result is computed by cosine intermediate result which uses the sine intermediate result als threshold to decide when angles must be enter the third quadrant of the unit circle.

$$f_{ang} = \begin{cases} f_c & f_s \geq 0 \\ -f_c + 2\pi & f_s < 0 \end{cases}$$

A the second angle reconstruction can be achieved by the atan2 function which has although an interval shift at 180 degree. It implies to use the sine results as threshold too.

$$f_{ang} = \begin{cases} f_a & f_s \geq 0 \\ f_a + 2\pi & f_s < 0 \end{cases}$$

## Syntax

```
[fang, fc, fs, fa] = sinoids2angles(fsin, fcos, frad, rad)
```

## Description

**[fang, fc, fs, fa] = sinoids2angles(fsin, fcos, frad, rad)** returns angles in rad or degrees given by corresponding sinoids and radius. Set rad flag to false if angles in degrees are needed.

## Examples

```
phi = linspace(0, 2*pi, 100);
fsin = sin(phi);
fcos = cos(phi);
frad = sqrt(fsin.^2 + fcos.^2);
[fang, fc, fs, fa] = sinoids2angles(fsin, fcos, frad, true)
```

## Input Argurments

**fsin** is a scalar or vector with sine information to corresponding angle.

**fcos** is a scalar or vector with cosine information to corresponding angle.

**frad** is a scalar or vector which represents the radius of each sine and cosine position.

**rad** is a boolean flag. If it is false the resulting angles are converted into degrees. If it is true fang is returned in rad. Default is true.

**how** is char vector which gives option how to reconstruct angles via acos or atan2 function. Default is atan2. Both methods use asin function as threshold to switch 180° intervall.

## Output Argurments

**fang** is a scalar or vector with angles in rad or degree corresponding to the sine and cosine inputs.

**fc** is a scalar or vector with angles directly computed by cosine and radius.

**fs** is a scalar or vector with angles directly computed by sine and radius.

**fa** is a scalar or vector with angles directly computed by sine and cosine.

## Requirements

- Other m-files required: None
- Subfunctions: acos, asin
- MAT-files required: None

## See Also

- angles2sinoids

Created on December 31. 2020 by Tobias Wulf. Copyright Tobias Wulf 2020.

```
function [fang, fc, fs, fa] = sinoids2angles(fsin, fcos, frad, rad, how)
    arguments
        % validate sinoids and radius as scalar or vector of the same size
        fsin (:,1) double {mustBeReal}
        fcos (:,1) double {mustBeReal, mustBeEqualSize(fsin, fcos)}
        frad (:,1) double {mustBeReal, mustBeEqualSize(fsin, frad)}
        % validate rad as boolean flag with default true
        rad (1,1) logical {mustBeNumericOrLogical} = true
        % validate how as char option flag with default atan2
        how (1,:) char {mustBeText} = 'atan2'
    end

    % compute angles by cosine, sine and radius
    fc = acos(fcos ./ frad);
    fs = asin(fsin ./ frad);
    fa = atan2(fsin, fcos);

    % get indices for interval > 180°
    idx = fs < 0;

    switch how
        case 'acos'
            % angles from cosine
            fang = fc;

            % correct 180° interval
            fang(idx) = -1 * fang(idx) + 2 * pi;
        case 'atan2'
            fang = fa;

            % correct 180° interval
            fang(idx) = fang(idx) + 2 * pi;

        otherwise
            error('Unknow arc function for reconstruction: %s.', how)
    end

    % return degrees if not rad
    if ~rad, fang = 180 / pi * fang; end
end

% Custom validation function
function mustBeEqualSize(a,b)
    % Test for equal size
    if ~isequal(size(a),size(b))
        eid = 'Size:notEqual';
        msg = 'Size of first input must equal size of second input.';
        throwAsCaller(MException(eid,msg))
    end
end
```