2021/7/26 tutorial

```
1 #= ====
             ***** tutorial to show how it works *****
                                                            ==== =#
 2 using PyCall
 3 using PyPlot
 4 using LinearAlgebra
 5 using Polynomials
 7 ts = sinpi.(0:0.2:10) .+ 0.01;
 8
9 M = 7; N = 5;
10 \mid TotN = N + M;
11 # N > rank(M) will be required to obtain stable solution
12 #
        to avoid singularity by degeneration
13
14 figure();
15 plot(ts, label="whole time series sin(2\pi ft)+0.01");
16 plot(ts[1:TotN], label="$TotN samples for single analysis");
17 legend(); show();
18
19 myeps = 1e-15;
20
21 # ----- internal functions;
22 """find extreme value in matrix A""";
23 extreme(A<sub>ij</sub>) =
      A<sub>ij</sub> |> maximum |> abs > A<sub>ij</sub> |> minimum |> abs ? maximum(A<sub>ij</sub>) : minimum(A<sub>ij</sub>);
25 """find exterme value in vector x""";
26 absmax(x) = maximum(abs.(extrema(x)));
27 """corresponding index of above vector x""";
28 \mid absmaxidx(x) = filter(i -> abs(x[i]) == absmax(x), eachindex(x));
29
30 # ----- set autocorrelation eq. AX=B
31 A = zeros(M, M);
32 for m in 1:M, m' in 1:M, n in 0:N - 1
       A[m,m'] += ts[TotN - m - n] * ts[TotN - m' - n]
34 end
35 A
36
37 B = zeros(M);
38 for m' in 1:M, n in 0:N - 1
       B[m'] += ts[TotN - 0 - n] * ts[TotN - m' - n]
39
40 end
41 B'
42
43|# ----- normalize matrix
44 scaler = extreme(A);
45 A /= scaler;
46 B /= scaler;
47 A
48 B'
49
50 # ----- care for void records: safety reason
51 for i in 1:M
52
       if norm(A[i,:]) < myeps</pre>
53
          A[i,:] = zeros(M)
           B[i] = 0
54
           A[i,i] = 1
55
56
       end
57 end
58 A
59 B'
60
```

localhost:4649/?mode=julia 1/3

2021/7/26 tutoria

```
61 # ----- pivotting
62 for i in 1:M - 1
       amidx = absmaxidx(A[i:M,i])[1] + i - 1
63
       A[amidx,:], A[i,:] = A[i,:], A[amidx,:]
64
65
       B[amidx], B[i] = B[i], B[amidx]
66 end
67 A
68 B'
69
70 # ----- sweepout forward
71 for i in 1:M - 1
72
       if abs(A[i,i]) < myeps</pre>
73
           A[i,:] = zeros(M)
74
           B[i] = 0
75
           A[i,i] = 1
76
       end
77
       for j = i + 1:M
78
           mx = A[j,i] / A[i,i]
79
           A[j,:] .-= mx * A[i,:]
80
           B[j] -= mx * B[i]
81
       end
82 end
83 A
84 B'
85
86 # ----- sweepout backward
87 for i in M:-1:2
88
       if abs(A[i,i]) < myeps</pre>
89
           A[i,:] = zeros(M)
90
           B[i] = 0
91
           A[i,i] = 1
92
       end
93
       for j = 1:i - 1
94
           mx = A[j,i] / A[i,i]
95
           A[j,:] .-= mx * A[i,:]
96
           B[j] -= mx * B[i]
97
       end
       B[i] /= A[i,i]
98
99
       A[i,:] /= A[i,i]
100 end
101 | B[1] /= A[1,1];
102 A[1,:] /= A[1,1];
103 A
104 B'
105
106 \# ----- remove null higher orders a_m (m>M')
107 M' = M;
108 for i in 1:M
       if abs(B[i]) > myeps
109
110
           M' = i
111
       end
112 end
113 M'
114
115 # ----- set prediction coeffs. & get modes
116 predcoeffs = ones(M' + 1);
117 for i in 1:M'
118
       predcoeffs[i] = -B[M' + 1 - i]
119 end
120 predcoeffs'
```

localhost:4649/?mode=julia 2/3

2021/7/26 tutorial.jl

```
121 Polynomial(predcoeffs)
122 modes = roots(Polynomial(predcoeffs))
124 # ----- remove exterme modes which corrrespond to noise floor
125 MaxDiffBetweenEdges = 100;
126 M'' = 0;
127 for i in 1:M'
128
        growwidth = abs(modes[i])^TotN
129
        if maximum([growwidth, 1 / growwidth]) < MaxDiffBetweenEdges</pre>
            M'' += 1
130
            modes[M''] = modes[i]
131
132
        end
133 end
134 modes = modes[1:M'']
135 M''
136
137 # ----- calc complex amps. at left bound solve AX=B
138 A = zeros(ComplexF64, M'', M'');
139 B = zeros(ComplexF64, M'');
140
141 for i in 1:M'', j in 1:M'', n in 0:TotN - 1
        A[i,j] += (modes[i] * modes[j])^n
143 end
144 A
145
146 for i in 1:M'', n in 0:TotN - 1
147
       B[i] += ts[n + 1] * modes[i]^n
148 end
149 B
150
151 | iCAmp = A \setminus B
152
153 # ----- prepare return values
154 iFrq = imag(log.(modes)) / 2\pi;
155 iAVR = real(log.(modes));
156 results = [];
157 for i in 1:M''
        push!(results, (iFrq = iFrq[i], iAVR = iAVR[i], iCAmp = iCAmp[i]))
158
159 end
160 results
161
162 # ----- plot spectrum of each mode
163 """ equation for Lorentz profile spectrum """;
164 LorentzProf(f, iFrq, iAVR, iCAmp) =
165
        abs(iCAmp) * \sqrt{(iAVR^2 / ((2\pi * (abs(iFrq) - f))^2 + iAVR^2))};
166
167 figure();
168 \times = 0:1e-4:0.2;
169 for i in 1:M''
       y = LorentzProf.(x, iFrq[i],
                                             iAVR[i],
                                                              iCAmp[i]
170 #
       y = LorentzProf.(x, results[i][1], results[i][2], results[i][3]
171 #
172
       y = LorentzProf.(x, results[i].iFrq, results[i].iAVR, results[i].iCAmp);
        plot(x, y, label="mode no. $i");
173
174 end
175 yscale("log"); legend(); show();
176
```

localhost:4649/?mode=julia 3/3