```julia
#= ====     ***** tutorial to show how it works *****     ==== =#
using PyCall
using PyPlot
using LinearAlgebra
using Polynomials

ts = sinpi.(0:0.2:10) .+ 0.01;

M = 7; N = 5;
TotN = N + M;
# N > rank(M) will be required to obtain stable solution
#     to avoid singularity by degeneration

figure();
plot(ts, label="whole time series sin(2πft)+0.01");
plot(ts[1:TotN], label="$TotN samples for single analysis");
legend(); show();

myeps = 1e-15;

# ----------------------------------- internal functions;
"""find extreme value in matrix A""";
extreme(Aᵢⱼ) =
    Aᵢⱼ |> maximum |> abs > Aᵢⱼ |> minimum |> abs ? maximum(Aᵢⱼ) : minimum(Aᵢⱼ);
"""find exterme value in vector x""";
absmax(x) = maximum(abs.(extrema(x)));
"""corresponding index of above vector x""";
absmaxidx(x) = filter(i -> abs(x[i]) == absmax(x), eachindex(x));

# ----------------------------- set autocorrelation eq. AX=B
A = zeros(M, M);
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]
end
A

B = zeros(M);
for m′ in 1:M, n in 0:N - 1
    B[m′] += ts[TotN - 0 - n] * ts[TotN - m′ - n]
end
B'

# ------------------------------------ normalize matrix
scaler = extreme(A);
A /= scaler;
B /= scaler;
A
B'

# ----------------- care for void records: safety reason
for i in 1:M
    if norm(A[i,:]) < myeps
        A[i,:] = zeros(M)
        B[i] = 0
        A[i,i] = 1
    end
end
A
B'

```

```julia
# ------------------------------------ pivotting
for i in 1:M - 1
    amidx = absmaxidx(A[i:M,i])[1] + i - 1
    A[amidx,:], A[i,:] = A[i,:], A[amidx,:]
    B[amidx], B[i] = B[i], B[amidx]
end
A
B'

# ----------------------------------- sweepout forward
for i in 1:M - 1
    if abs(A[i,i]) < myeps
        A[i,:] = zeros(M)
        B[i] = 0
        A[i,i] = 1
    end
    for j = i + 1:M
        mx = A[j,i] / A[i,i]
        A[j,:] .-= mx * A[i,:]
        B[j] -= mx * B[i]
    end
end
A
B'

# --------------------------------- sweepout backward
for i in M:-1:2
    if abs(A[i,i]) < myeps
        A[i,:] = zeros(M)
        B[i] = 0
        A[i,i] = 1
    end
    for j = 1:i - 1
        mx = A[j,i] / A[i,i]
        A[j,:] .-= mx * A[i,:]
        B[j] -= mx * B[i]
    end
    B[i] /= A[i,i]
    A[i,:] /= A[i,i]
end
B[1] /= A[1,1];
A[1,:] /= A[1,1];
A
B'

# ------------------------- remove null higher orders aₘ (m>M′)
M′ = M;
for i in 1:M
    if abs(B[i]) > myeps
        M′ = i
    end
end
M′

# ------------------------------ set prediction coeffs. & get modes
predcoeffs = ones(M′ + 1);
for i in 1:M′
    predcoeffs[i] = -B[M′ + 1 - i]
end
predcoeffs'
```

```julia
Polynomial(predcoeffs)
modes = roots(Polynomial(predcoeffs))

# ------------ remove exterme modes which corrrespond to noise floor
MaxDiffBetweenEdges = 100;
M′′ = 0;
for i in 1:M′
    growwidth = abs(modes[i])^TotN
    if maximum([growwidth, 1 / growwidth]) < MaxDiffBetweenEdges
        M′′ += 1
        modes[M′′] = modes[i]
    end
end
modes = modes[1:M′′]
M′′

# ---------- calc complex amps. at left bound solve AX=B
A = zeros(ComplexF64, M′′, M′′);
B = zeros(ComplexF64, M′′);

for i in 1:M′′, j in 1:M′′, n in 0:TotN - 1
    A[i,j]  += (modes[i] * modes[j])^n
end
A

for i in 1:M′′, n in 0:TotN - 1
    B[i] += ts[n + 1] * modes[i]^n
end
B

iCAmp = A \ B

# -------------------  prepare return values
iFrq = imag(log.(modes)) / 2π;
iAVR = real(log.(modes));
results = [];
for i in 1:M′′
    push!(results, (iFrq = iFrq[i], iAVR = iAVR[i], iCAmp = iCAmp[i]))
end
results

# ------------------- plot spectrum of each mode
""" equation for Lorentz profile spectrum """;
LorentzProf(f,iFrq,iAVR,iCAmp) =
    abs(iCAmp) * √(iAVR^2 / ((2π * (abs(iFrq) - f))^2 + iAVR^2));

figure();
x = 0:1e-4:0.2;
for i in 1:M′′
#    y = LorentzProf.(x, iFrq[i],         iAVR[i],         iCAmp[i]        );
#    y = LorentzProf.(x, results[i][1],   results[i][2],   results[i][3]   );
    y = LorentzProf.(x, results[i].iFrq, results[i].iAVR, results[i].iCAmp);
    plot(x, y, label="mode no. $i");
end
yscale("log"); legend(); show();
```