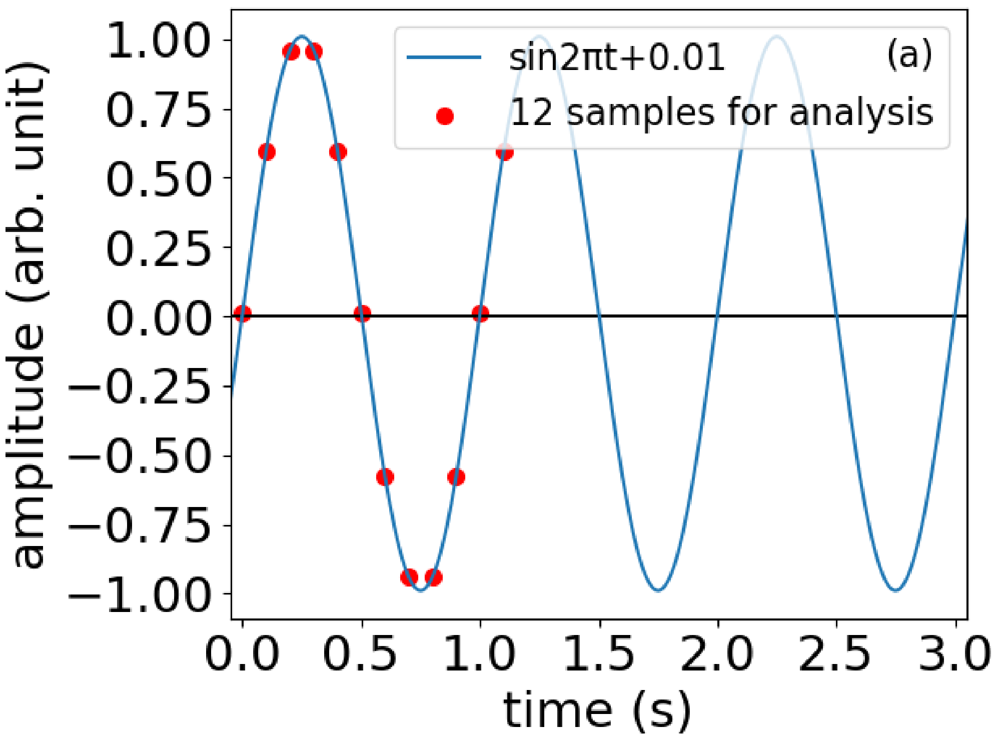
```
julia> #= ====        ***** tutorial to show how it works *****        ==== =#
julia> using PyCall
julia> using PyPlot
julia> using LinearAlgebra
julia> using Polynomials
julia> using FFTW
julia>
julia> t=0:0.1:3;
julia> t2=-1:0.01:4;
julia>
julia> fnts(t)=sinpi.(2t) .+ 0.01;
julia> ts = fnts(t);
julia> ts2= fnts(t2);
julia>
julia> M = 7; N = 5;
julia> TotN = N + M;
julia> # N > rank(M) will be required to obtain stable solution
julia> #        to avoid singularity by degeneration
julia>
julia> fs=22;
julia> figure(tight_layout=true);
julia> xlim(-0.05,3.05);
julia> plot([-0.1,3.1],[0,0],label="", color="black");
julia> plot(t2,ts2, label="sin2 π t+0.01");
julia> scatter(t[1:TotN], ts[1:TotN], label="$TotN samples for analysis", color="red", lw=2);
julia> xlabel("time (s)",fontsize=fs);
julia> ylabel("amplitude (arb. unit)",fontsize=fs);
julia> xticks(fontsize=fs); yticks(fontsize=fs);
julia> legend(fontsize=16, loc="upper right");
julia> annotate("(a)",(2.7,0.9),fontsize=16,zorder=6);
julia> show();
```



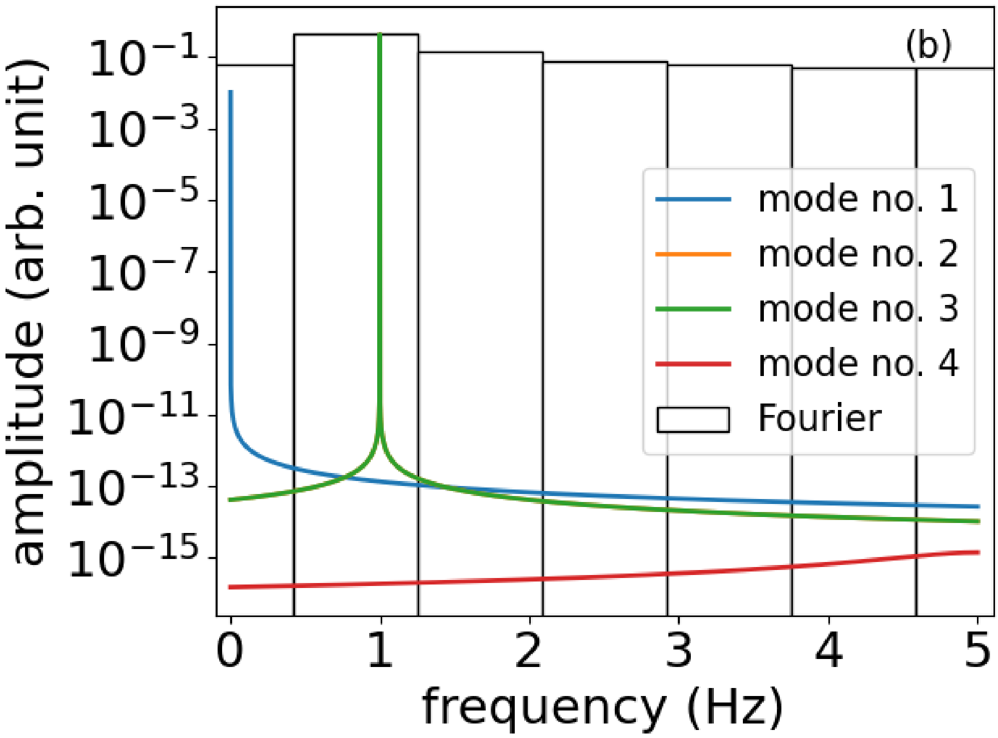
```
julia>
julia> myeps = 1e-15;
julia>
julia> # ---------------------------------- internal functions;
julia> """find extreme value in matrix A""";
julia> extreme(Aᵢⱼ) =
              Aᵢⱼ |> maximum |> abs > Aᵢⱼ |> minimum |> abs ? maximum(Aᵢⱼ) : minimum(Aᵢⱼ);
julia> """find exterme value in vector x""";
julia> absmax(x) = maximum(abs.(extrema(x)));
julia> """corresponding index of above vector x""";
julia> absmaxidx(x) = filter(i -> abs(x[i]) == absmax(x), eachindex(x));
julia>
julia> # ---------------------------- set autocorrelation eq. AX=B
julia> A = zeros(M, M);
julia> 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
julia> A
7×7 Matrix{Float64}:
  2.43895    1.96149    0.723245  -0.802819  -2.0338    -2.4995    -2.02204
  1.96149    2.43895    1.97324    0.742266  -0.783798  -2.02204   -2.4995
  0.723245   1.97324    2.46246    2.00402    0.773042  -0.760287  -2.01029
 -0.802819   0.742266   2.00402    2.5005     2.04206    0.803819  -0.741266
 -2.0338    -0.783798   0.773042   2.04206    2.53854    2.07284    0.82284
 -2.4995    -2.02204   -0.760287   0.803819   2.07284    2.56205    2.0846
 -2.02204   -2.4995    -2.01029   -0.741266   0.82284    2.0846     2.56205
julia>
julia> B = zeros(M);
julia> for m′ in 1:M, n in 0:N - 1
              B[m′] += ts[TotN - 0 - n] * ts[TotN - m′ - n]
           end
julia> B'
1×7 adjoint(::Vector{Float64}) with eltype Float64:
 1.97324  0.723245  -0.810085  -2.04106  -2.4995  -2.01029  -0.760287
julia>
julia> # ----------------------------------- normalize matrix
julia> scaler = extreme(A);
julia> A /= scaler;
julia> B /= scaler;
julia> A
7×7 Matrix{Float64}:
  0.95195    0.765592   0.282291  -0.31335   -0.793816  -0.975585  -0.789227
  0.765592   0.95195    0.770181   0.289715  -0.305926  -0.789227  -0.975585
  0.282291   0.770181   0.961127   0.782193   0.301728  -0.296749  -0.784639
 -0.31335    0.289715   0.782193   0.975975   0.797042   0.31374   -0.289325
 -0.793816  -0.305926   0.301728   0.797042   0.990823   0.809054   0.321164
 -0.975585  -0.789227  -0.296749   0.31374    0.809054   1.0        0.813643
 -0.789227  -0.975585  -0.784639  -0.289325   0.321164   0.813643   1.0
julia> B'
1×7 adjoint(::Vector{Float64}) with eltype Float64:
 0.770181  0.282291  -0.316186  -0.796651  -0.975585  -0.784639  -0.296749
julia>
```

```
julia> # ---------------- care for void records: safety reason
julia> for i in 1:M
           if norm(A[i,:]) < myeps
               A[i,:] = zeros(M)
                   B[i] = 0
               A[i,i] = 1
           end
       end
julia> A
7×7 Matrix{Float64}:
  0.95195    0.765592   0.282291  -0.31335   -0.793816  -0.975585  -0.789227
  0.765592   0.95195    0.770181   0.289715  -0.305926  -0.789227  -0.975585
  0.282291   0.770181   0.961127   0.782193   0.301728  -0.296749  -0.784639
 -0.31335    0.289715   0.782193   0.975975   0.797042   0.31374   -0.289325
 -0.793816  -0.305926   0.301728   0.797042   0.990823   0.809054   0.321164
 -0.975585  -0.789227  -0.296749   0.31374    0.809054   1.0        0.813643
 -0.789227  -0.975585  -0.784639  -0.289325   0.321164   0.813643   1.0
julia> B'
1×7 adjoint(::Vector{Float64}) with eltype Float64:
 0.770181  0.282291  -0.316186  -0.796651  -0.975585  -0.784639  -0.296749
julia>
julia> # ----------------------------------- pivotting
julia> 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
julia> A
7×7 Matrix{Float64}:
 -0.975585  -0.789227  -0.296749   0.31374    0.809054   1.0        0.813643
 -0.789227  -0.975585  -0.784639  -0.289325   0.321164   0.813643   1.0
  0.282291   0.770181   0.961127   0.782193   0.301728  -0.296749  -0.784639
 -0.31335    0.289715   0.782193   0.975975   0.797042   0.31374   -0.289325
 -0.793816  -0.305926   0.301728   0.797042   0.990823   0.809054   0.321164
  0.95195    0.765592   0.282291  -0.31335   -0.793816  -0.975585  -0.789227
  0.765592   0.95195    0.770181   0.289715  -0.305926  -0.789227  -0.975585
julia> B'
1×7 adjoint(::Vector{Float64}) with eltype Float64:
 -0.784639  -0.296749  -0.316186  -0.796651  -0.975585  0.770181  0.282291
julia>
julia> # ---------------------------------- sweepout forward
julia> 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
```

```
julia> A
7×7 Matrix{Float64}:
 -0.975585  -0.789227  -0.296749     0.31374     0.809054     1.0          0.813643
  0.0       -0.337116  -0.544575    -0.543134   -0.333343     0.00466387   0.34178
  0.0        0.0        1.95713e-5   5.12382e-5  8.29052e-5   0.000102476  0.000102476
  0.0        0.0        0.0          1.0         0.0          0.0          0.0
  0.0        0.0        0.0          0.0         1.52344e-15  2.42994e-15  2.73085e-15
  0.0        0.0        0.0          0.0         0.0          1.0          0.0
  0.0        0.0       -3.38813e-21  0.0         0.0          0.0         -9.40416e-16
julia> B'
1×7 adjoint(::Vector{Float64}) with eltype Float64:
 -0.784639  0.338007  1.95713e-5  0.0  1.1226e-15  0.0  -9.15407e-16
julia>
julia> # -------------------------------- sweepout backward
julia> 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
julia> B[1] /= A[1,1];
julia> A[1,:] /= A[1,1];
julia> A
7×7 Matrix{Float64}:
  1.0  -0.0  -0.0  -0.0  -0.0  -0.0  -0.0
 -0.0   1.0  -0.0  -0.0  -0.0  -0.0  -0.0
  0.0   0.0   1.0   0.0   0.0   0.0   0.0
  0.0   0.0   0.0   1.0   0.0   0.0   0.0
  0.0   0.0   0.0   0.0   1.0   0.0   0.0
  0.0   0.0   0.0   0.0   0.0   1.0   0.0
  0.0   0.0   0.0   0.0   0.0   0.0   1.0
julia> B'
1×7 adjoint(::Vector{Float64}) with eltype Float64:
 0.688853  1.69575  -2.12148  0.0  0.736882  0.0  0.0
julia>
julia> # ------------------------ remove null higher orders a_m (m>M′ )
julia> M′  = M;
julia> for i in 1:M
           if abs(B[i]) > myeps
               M′  = i
           end
       end
julia> M′
5
julia>
```

```
julia> # ------------------------------ set prediction coeffs. & get modes
julia> predcoeffs = ones(M′ + 1);
julia> for i in 1:M′
          predcoeffs[i] = -B[M′ + 1 - i]
       end
julia> predcoeffs'
1×6 adjoint(::Vector{Float64}) with eltype Float64:
 -0.736882  -0.0  2.12148  -1.69575  -0.688853  1.0
julia> Polynomial(predcoeffs)
Polynomial(-0.7368817567792972 + 2.121481213065832*x^2 - 1.6957469524556559*x^3 - 0.6888525038427817*x^4 + 1.0*x^5)
julia> modes = roots(Polynomial(predcoeffs))
5-element Vector{ComplexF64}:
 -1.4045379479974356 + 0.0im
 -0.5246435369181329 + 0.0im
  0.8090169943749257 - 0.5877852522924136im
  0.8090169943749257 + 0.5877852522924136im
   1.000000000008501 + 0.0im
julia>
julia> # ------------ remove exterme modes which corrrespond to noise floor
julia> MaxDiffBetweenEdges = 100;
julia> M′′ = 0;
julia> for i in 1:M′
          growwidth = abs(modes[i])^TotN
             if maximum([growwidth, 1 / growwidth]) < MaxDiffBetweenEdges
                   M′′ += 1
             modes[M′′] = modes[i]
          end
       end
julia> modes = modes[1:M′′]
4-element Vector{ComplexF64}:
 -1.4045379479974356 + 0.0im
  0.8090169943749257 - 0.5877852522924136im
  0.8090169943749257 + 0.5877852522924136im
   1.000000000008501 + 0.0im
julia> M′′
4
julia>
julia> # ---------- calc complex amps. at left bound solve AX=B
julia> A = zeros(ComplexF64, M′′, M′′);
julia> B = zeros(ComplexF64, M′′);
julia>
julia> for i in 1:M′′, j in 1:M′′, n in 0:TotN - 1
          A[i,j] += (modes[i] * modes[j])^n
          end
julia> A
4×4 Matrix{ComplexF64}:
  3570.16+0.0im       -15.8329+20.1204im  -15.8329-20.1204im  -24.0956+0.0im
 -15.8329+20.1204im   1.30902-0.951057im      12.0+0.0im      1.80902-0.587785im
 -15.8329-20.1204im       12.0+0.0im      1.30902+0.951057im  1.80902+0.587785im
 -24.0956+0.0im       1.80902-0.587785im  1.80902+0.587785im      12.0+0.0im
julia>
julia> for i in 1:M′′, n in 0:TotN - 1
          B[i] += ts[n + 1] * modes[i]^n
          End
```

```
julia> B
4-element Vector{ComplexF64}:
  -20.361325254376972 + 0.0im
  0.49361842809224077 - 5.351369355333874im
  0.49361842809224077 + 5.351369355333874im
    0.7077852522222327 + 0.0im
julia>
julia> iCAmp = A ¥ B
4-element Vector{ComplexF64}:
  1.4140565131219918e-15 + 2.487784153979577e-19im
    7.051881123245964e-14 + 0.5000000000002429im
    7.052191153141293e-14 - 0.5000000000002429im
      0.009999999999492205 - 6.100912414782267e-21im
julia>
julia> # -------------------    prepare return values
julia> iFrq = imag(log.(modes)) / 2 π ;
julia> iAVR = real(log.(modes));
julia> results = [];
julia> for i in 1:M′ ′
                push!(results, (iFrq = iFrq[i], iAVR = iAVR[i], iCAmp = iCAmp[i]))
                end
julia> results
4-element Vector{Any}:
  (iFrq = 0.5, iAVR = 0.339708386063257, iCAmp = 1.4140565131219918e-15 + 2.487784153979577e-19im)
  (iFrq = -0.09999999999999439, iAVR = -5.2569060216003926e-14, iCAmp = 7.051881123245964e-14 + 0.5000000000002429im)
  (iFrq = 0.09999999999999439, iAVR = -5.2569060216003926e-14, iCAmp = 7.052191153141293e-14 - 0.5000000000002429im)
  (iFrq = 0.0, iAVR = 8.50097769951869e-12, iCAmp = 0.009999999999492205 - 6.100912414782267e-21im)
julia> results=reverse(results);
julia>
julia> # ------------------- plot spectrum of each mode
julia> """ equation for Lorentz profile spectrum """;
julia> LorentzProf(f,iFrq,iAVR,iCAmp) =
                abs(iCAmp) * √(iAVR^2 / ((2 π  * (abs(iFrq) - f))^2 + iAVR^2));
julia>
julia>        figure(tight_layout=true);
julia>        #--------------------- FFT for demo
julia>        yFFT=abs.(fft(ts[1:TotN]))[1:7]/TotN;
julia>        xFFT=0:1:6;
julia>        bar(xFFT/1.2,yFFT,width=1/1.2,color="#FFFFFF",edgecolor="#000000",label="Fourier");
julia>        #---------------------------
julia>        x = 0:1e-4:0.5;
julia>        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(10x, y, label="mode no. $i", lw=2);
                end
julia>        annotate("(b)",(4.5,0.1),fontsize=16);
julia>        xlabel("frequency (Hz)", fontsize=fs);
julia>        ylabel("amplitude (arb. unit)", fontsize=fs);
julia>        xticks(fontsize=fs); yticks(fontsize=fs);
julia>        xlim(-0.1,5.1);
julia>        yscale("log"); legend(fontsize=16, loc="center right"); show();
```



(b)
amplitude (arb. unit)
frequency (Hz)
mode no. 1
mode no. 2
mode no. 3
mode no. 4
Fourier