```
SetDirectory @ NotebookDirectory[];
Import["https://qtechtheory.org/QuESTlink.m"];
CreateDownloadedQuESTEnv[];
```

```
freqs = {-0.7098301012032957`, -0.05177915010485945`,
   0.9065658350752286`, -0.9265872629174545`, 0.09500423337423802`,
   -0.4959779888875864`, -0.16859651542448084`, 0.8678962526289684`,
   -0.11732544489533625`, 0.907000173126657`, -0.44233758592927064`,
   0.5741970506048069`, 0.06929812708367722`, 0.23796927148578906`,
   -0.9361278713972463`, 0.4908133930451908`, 0.9021460849017751`,
   0.7153291490064708`, -0.309302916408313`, -0.38233264968765734`};
```

```
spinChainHamil[J_, numQs_] := Module[{hamil},
   hamil = ExpandAll@Sum[J * (X_k X_{k+1} + Y_k Y_{k+1} + Z_k Z_{k+1}), {k, 0, numQs - 2}] +
     ExpandAll[J * (X_0 X_{numQs-1} + Y_0 Y_{numQs-1} + Z_0 Z_{numQs-1})];
   hamil += Sum[freqs[[k]] * Z_{k-1}, {k, 1, numQs}];
   Return@hamil;
  ];
```

calculate symbolic Hamiltonian

```
numQs = 6;
hamil = Chop@spinChainHamil[.1, numQs]
```

$0.1\, X_0 X_1 + 0.1\, X_1 X_2 + 0.1\, X_2 X_3 + 0.1\, X_3 X_4 + 0.1\, X_0 X_5 + 0.1\, X_4 X_5 + 0.1\, Y_0 Y_1 + 0.1\, Y_1 Y_2 + 0.1\, Y_2 Y_3 + 0.1\, Y_3 Y_4 + 0.1\, Y_0 Y_5 + 0.1\, Y_4 Y_5 - 0.70983\, Z_0 - 0.0517792\, Z_1 + 0.1\, Z_0 Z_1 + 0.906566\, Z_2 + 0.1\, Z_1 Z_2 - 0.926587\, Z_3 + 0.1\, Z_2 Z_3 + 0.0950042\, Z_4 + 0.1\, Z_3 Z_4 - 0.495978\, Z_5 + 0.1\, Z_0 Z_5 + 0.1\, Z_4 Z_5$

code for generating all 3-local Paulis

```
pauli[type_, site_] := type_site
getListOf3LocalPaulis[numQs_] := Join[
  Flatten[Table[pauli[type, site], {site, 0, numQs - 1}, {type, {X, Y, Z}}], 1]
  ,
  Flatten[
   Table[pauli[type1, site1] × pauli[type2, site2], {site1, 0, numQs - 1},
    {site2, site1 + 1, numQs - 1}, {type1, {X, Y, Z}}, {type2, {X, Y, Z}}]
   , 3]
  ,
  Flatten[
   Table[pauli[type1, site1] × pauli[type2, site2] × pauli[type3, site3],
    {site1, 0, numQs - 1}, {site2, site1 + 1, numQs - 1}, {site3, site2 + 1, numQs - 1},
    {type1, {X, Y, Z}}, {type2, {X, Y, Z}}, {type3, {X, Y, Z}}]
   , 5]
 ]

getListOf2LocalPaulis[numQs_] := Join[
  Flatten[Table[{pauli[type, site]}, {site, 0, numQs - 1}, {type, {X, Y, Z}}], 1]
  ,
  Flatten[
   Table[{pauli[type1, site1], pauli[type2, site2]}, {site1, 0, numQs - 1},
    {site2, site1 + 1, numQs - 1}, {type1, {X, Y, Z}}, {type2, {X, Y, Z}}]
   , 3]
 ]
Length@getListOf3LocalPaulis[numQs]
693
```

calculate Hamiltonian matrix and dt time-step propagator

```
Hmat = CalcPauliExpressionMatrix[hamil];
(* use all up to 3-local Paulis *)
listOfPaulis = getListOf3LocalPaulis[numQs];
eigs = Eigenvalues[Hmat, 4];
vecs = Eigenvectors[Hmat, 4];
dt = 3;
Umat = MatrixExp[-ⅈ * dt * Hmat];
ll = 200;(* number of total time steps *)
TT = ll * dt; (* total simulation time *)
Print["eigs:  ", eigs[[1 ;; 3]]]
```

```
eigs:  {-3.46538, -3.3858, 3.22729}
```

simulate dynamics and calculate expected values

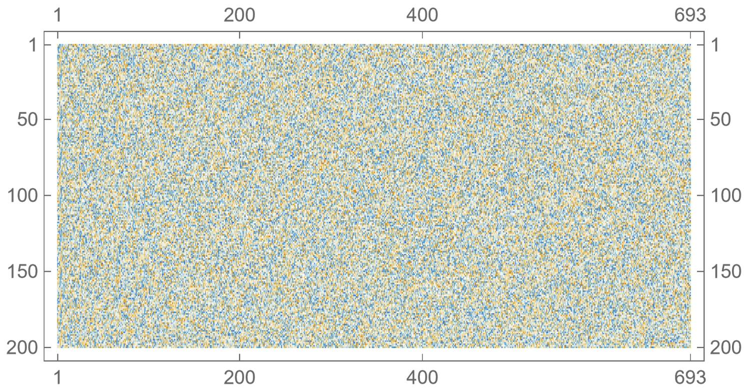
```
psinull = (vecs⟦1⟧ + 0.1 vecs⟦2⟧ + 0.1 vecs⟦3⟧);
psinull = psinull / Norm[psinull];
psi = psinull;
{ψ, ϕ} = CreateQuregs[numQs, 2];
observables = {};
observablesSHADOW = {};
Do[
 psi = Umat.psi;
 SetQuregMatrix[ψ, psi];


 AppendTo[observablesSHADOW,
  (* these are built-in QuEST-link functions for simulating shadows *)
  data = SampleClassicalShadow[ψ, 1000];
  CalcExpecPauliProdsFromClassicalShadow[data, listOfPaulis, 3]
 ];

 If[Mod[k, ll / 50] == 0,
  Print["progress = ", N@k / ll];
  (*Export["observables.m",Compress@observables];*)
  Export["observablesSHADOW.m", Compress@observablesSHADOW];
 ];

 , {k, 1, ll}]
```

```
progress = 0.02
progress = 0.04
progress = 0.06
progress = 0.08
progress = 0.1
progress = 0.12
progress = 0.14
progress = 0.16
progress = 0.18
progress = 0.2
progress = 0.22
progress = 0.24
progress = 0.26
```

```
progress = 0.28
progress = 0.3
progress = 0.32
progress = 0.34
progress = 0.36
progress = 0.38
progress = 0.4
progress = 0.42
progress = 0.44
progress = 0.46
progress = 0.48
progress = 0.5
progress = 0.52
progress = 0.54
progress = 0.56
progress = 0.58
progress = 0.6
progress = 0.62
progress = 0.64
progress = 0.66
progress = 0.68
progress = 0.7
progress = 0.72
progress = 0.74
progress = 0.76
progress = 0.78
progress = 0.8
progress = 0.82
progress = 0.84
progress = 0.86
progress = 0.88
progress = 0.9
progress = 0.92
progress = 0.94
progress = 0.96
progress = 0.98
progress = 1.
```

plot data -- row index is time variable, column index is observable

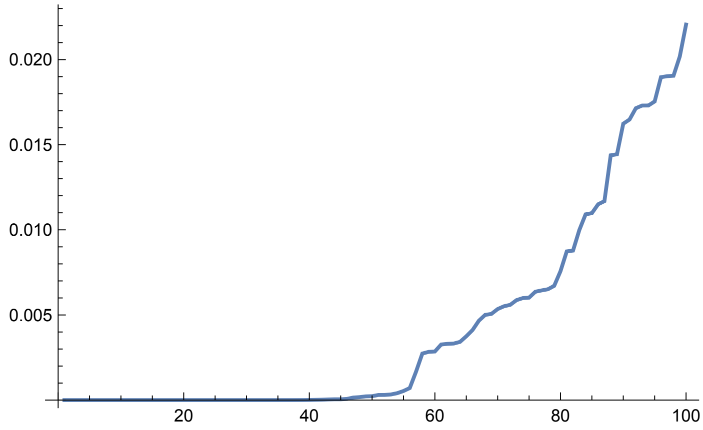
```
SetDirectory[NotebookDirectory[]];
observablesSHADOW = Uncompress@Import["observablesSHADOW.m"];
ll = Dimensions[observablesSHADOW][[1]];
dt = 3; TT = ll * dt;
observablesSHADOW = Transpose@observablesSHADOW;
(*  above is the non-
 square matrix that holds expected values as a function of time stepss  *)

observablesSHADOW = Standardize /@ observablesSHADOW;
plotD = Rasterize[MatrixPlot[Transpose@observablesSHADOW, AspectRatio → 1 / 2],
   RasterSize → {Automatic, 1000}, ImageSize → {Automatic, 200}]
(* With enough classical shadow data the column
 dimension should be much larger than the row dimension *)
```



perform a Ljung-box pre-screening of observables

```
dd = observablesSHADOW;
pValues = (AutocorrelationTest[#, ll - 1, "LjungBox"] &) /@ dd;
```

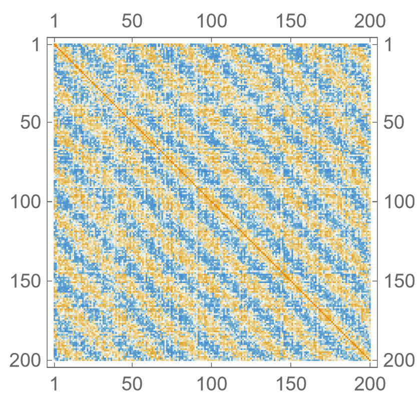
```
(* take only the best NMAX signals *)
NMAX = 100;
ListLinePlot[Sort[pValues]〚1 ;; NMAX〛, PlotRange → All]
pos = Ordering[pValues]〚1 ;; NMAX〛;
data = observablesSHADOW〚pos〛;
```



calculate the quadratic “covariance matrix”

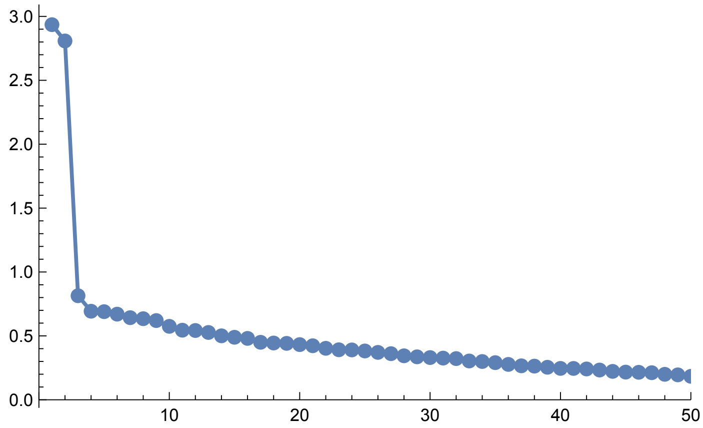
```
(*data=observablesSHADOW;*)
covariance = (Transpose[data].data) / (Length@observablesSHADOW);
Dimensions@covariance
plotC = Rasterize[MatrixPlot[covariance, AspectRatio → 1],
  RasterSize → {Automatic, 1000}, ImageSize → {Automatic, 200}]
```

```
{200, 200}
```



plot decaying eigenvalues of “covariance matrix” and calculate eigenvectors

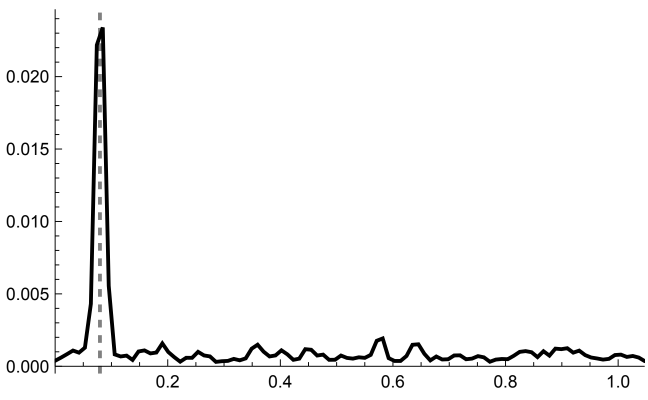
```
ee = Eigenvalues[covariance];
ListPlot[Chop@ee, PlotMarkers → Automatic,
 Joined → True, PlotRange → {{0, 50}, All}]
eigvs = Eigenvectors[covariance, 4];
```



Calculate spectral cross-correlation between the different eigenvectors

```
(* define window function *)
win = HannWindow[Range[-0.5, 0.5 - 1 / ll, 1 / ll]]

xcorr[x_, y_] := Module[{ll},
    (*see def at https://www.mathworks.com/help/matlab/ref/xcorr.html *)
    (*sx=Standardize[x];
    sy=Standardize[y];*)
    ll = Length@x;
    xc = Table[Mean[Drop[x, -m] * Drop[y, m]], {m, 1, ll - 1}];
    Return@xc;
  ];
llH = ll - 1; dt = 3; TTH = llH * dt;
win = HannWindow[Range[-0.5, 0.5 - 1 / llH, 1 / llH]];
data = Table[
    Fourier[win * xcorr[eigvs⟦k⟧, eigvs⟦l⟧]]
    , {k, 1, Length@eigvs}, {l, 1, Length@eigvs}];

solution = Table[
    aa = Chop@data⟦ ;; ,  ;; , k⟧;
    Max@SingularValueList[aa]
    , {k, 1, llH}];

spectrum = ListLinePlot[solution, PlotRange → {{0, 2 π * llH / (TTH) / 2}, All}
  , PlotStyle → Black
  , DataRange → {0, 2 π * llH / (TTH)}
  , GridLines → {{Abs[eigs⟦1⟧ - eigs⟦2⟧],
     Abs[eigs⟦1⟧ - eigs⟦3⟧],
     Abs[eigs⟦2⟧ - eigs⟦3⟧]}, None}
  , GridLinesStyle → Directive[Gray, Dashed, Thick]
  , ImageSize → {Automatic, 200}]

pos = Reverse[Ordering[Abs@solution]]⟦1⟧;
N@(llH - pos) / llH (2 π * llH / (TTH))
N@(pos) / llH (2 π * llH / (TTH))
Abs[eigs⟦1⟧ - eigs⟦2⟧]
Abs[eigs⟦1⟧ - eigs⟦3⟧]
```

{0., 0.00024672, 0.000986636, 0.00221902, 0.00394265, 0.00615583, 0.00885637, 0.0120416, 0.0157084, 0.0198532, 0.0244717, 0.0295596, 0.0351118, 0.0411227, 0.0475865, 0.0544967, 0.0618467, 0.069629, 0.077836, 0.0864597, 0.0954915, 0.104922, 0.114743, 0.124944, 0.135516, 0.146447, 0.157726, 0.169344, 0.181288, 0.193546, 0.206107, 0.218958, 0.232087, 0.245479, 0.259123, 0.273005, 0.28711, 0.301426, 0.315938, 0.330631, 0.345492, 0.360504, 0.375655, 0.390928, 0.406309, 0.421783, 0.437333, 0.452946, 0.468605, 0.484295, 0.5, 0.515705, 0.531395, 0.547054, 0.562667, 0.578217, 0.593691, 0.609072, 0.624345, 0.639496, 0.654508, 0.669369, 0.684062, 0.698574, 0.71289, 0.726995, 0.740877, 0.754521, 0.767913, 0.781042, 0.793893, 0.806454, 0.818712, 0.830656, 0.842274, 0.853553, 0.864484, 0.875056, 0.885257, 0.895078, 0.904508, 0.91354, 0.922164, 0.930371, 0.938153, 0.945503, 0.952414, 0.958877, 0.964888, 0.97044, 0.975528, 0.980147, 0.984292, 0.987958, 0.991144, 0.993844, 0.996057, 0.997781, 0.999013, 0.999753, 1., 0.999753, 0.999013, 0.997781, 0.996057, 0.993844, 0.991144, 0.987958, 0.984292, 0.980147, 0.975528, 0.97044, 0.964888, 0.958877, 0.952414, 0.945503, 0.938153, 0.930371, 0.922164, 0.91354, 0.904508, 0.895078, 0.885257, 0.875056, 0.864484, 0.853553, 0.842274, 0.830656, 0.818712, 0.806454, 0.793893, 0.781042, 0.767913, 0.754521, 0.740877, 0.726995, 0.71289, 0.698574, 0.684062, 0.669369, 0.654508, 0.639496, 0.624345, 0.609072, 0.593691, 0.578217, 0.562667, 0.547054, 0.531395, 0.515705, 0.5, 0.484295, 0.468605, 0.452946, 0.437333, 0.421783, 0.406309, 0.390928, 0.375655, 0.360504, 0.345492, 0.330631, 0.315938, 0.301426, 0.28711, 0.273005, 0.259123, 0.245479, 0.232087, 0.218958, 0.206107, 0.193546, 0.181288, 0.169344, 0.157726, 0.146447, 0.135516, 0.124944, 0.114743, 0.104922, 0.0954915, 0.0864597, 0.077836, 0.069629, 0.0618467, 0.0544967, 0.0475865, 0.0411227, 0.0351118, 0.0295596, 0.0244717, 0.0198532, 0.0157084, 0.0120416, 0.00885637, 0.00615583, 0.00394265, 0.00221902, 0.000986636, 0.00024672}



0.0736722

2.02072

0.0795835

6.69267