SampleExpecPauliString

```
SetDirectory @ NotebookDirectory[];
Import["../Link/QuESTlink.m"] // Quiet;
CreateLocalQuESTEnv["../quest_link"];
```

Doc

```
? SampleExpecPauliString
```

Symbol

SampleExpecPauliString[initQureg, channel, pauliString, numSamples] estimates the expected value of pauliString under the given channel (a circuit including decoherence) upon the state-vector initQureg, through Monte Carlo sampling. This avoids the quadratically greater memory costs of density-matrix simulation, but may need many samples to be accurate.

SampleExpecPauliString[initQureg, channel, pauliString, All] deterministically samples each channel decomposition once.

SampleExpecPauliString[initQureg, channel, pauliString, numSamples, {workQureg1, workQureg2}] uses the given persistent working registers to avoid their internal creation and destruction.

To get a sense of the circuits being sampled, see GetCircuitsFromChannel[].

Use option ShowProgress to monitor the progress of sampling.

```
? ShowProgress
```

Symbol

Optional argument to ApplyCircuit and SampleExpecPauliString, indicating whether to show a progress bar during circuit evaluation (default False). This slows evaluation slightly.

Correctness

```
n = 3;
{ψi, ϕ} = CreateQuregs[n, 2];
{ρ, ω} = CreateDensityQuregs[n, 2];

samps = {1, 10, 100, 1000};
```

```
getExactExpec[circ_, h_, ψi_] := (
    InitPureState[ρ, ψi];
    ApplyCircuit[ρ, circ];
    CalcExpecPauliString[ρ, h, ω]
)
```

```
setRandomState[ψi_] :=
    SetQuregMatrix[ψi, RandomComplex[{0,1+i}, 2^n]];
```

Deterministic

```
testDeterm[circ_] := Module[
    {h, eTrue, eDeterm, err},

    h = GetRandomPauliString[n];
    setRandomState[ψi];

    eTrue = getExactExpec[circ, h, ψi];
    eDeterm = SampleExpecPauliString[ψi, circ, h, All];
    err = (eTrue - eDeterm)/eTrue // Abs // Chop;

    Echo[eDeterm, "expec value: "];
    Echo[err, "error: "];
    If[err =!= 0, Style["ERRONEOUS EVALUATION", Red]]
]
```

Unitary mixtures

```
testDeterm @ Deph0[.1]
```

» expec value: -5.71611

» error: 0

```
testDeterm @ Deph0,1[.5]
```

» expec value: -13.9255

» error: 0

```
testDeterm @ Depol0[.5]
```

» expec value: -1.38894

» error: 0

```
testDeterm @ Depol0,2[.9]
```

» expec value: -5.28162

» error: 0

Non-unitary mixtures

```
testDeterm @ Damp0[.9]
```

» expec value: -2.27194

» error: 0

```
k = Table[RandomVariate @ CircularUnitaryMatrixDistribution[2], 3];
testDeterm @ KrausNonTP0[k]
```

» expec value: 8.22717

» error: 0

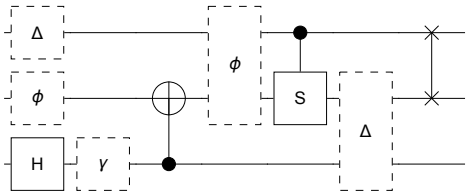
```
k = Table[RandomComplex[{-1 - ⅈ, 1 + ⅈ}, {2^2, 2^2}], 15];
testDeterm @ KrausNonTP0,1[k]
```

» expec value: -81.9149

» error: 0

Circuits

```
u = Circuit[
   H0 Damp0[.1] Deph1[.5] Depol2[.5] C0[X1] Deph1,2[.5] C2[S1] Depol0,1[.7] SWAP1,2];
DrawCircuit[u]
```




```
testDeterm @ u
```

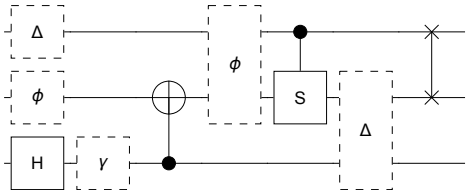
» expec value: -5.43185

» error: 0

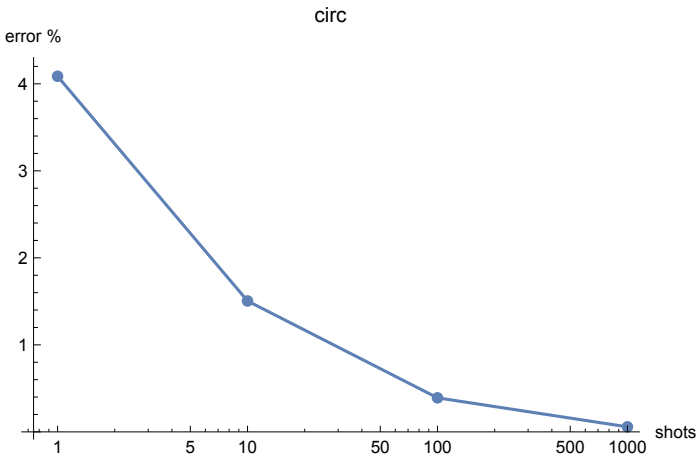
Monte Carlo

```
plotSampleErrors[circ_, shots_List, label_:None] := Module[
    {h, eTrue, eSamps, errs, data},

    h = GetRandomPauliString[n];
    setRandomState[ψi];

    eTrue = getExactExpec[circ, h, ψi];
    eSamps = Table[
        SampleExpecPauliString[ψi, circ, h, s],
        {s,shots}];
    errs = (eSamps - eTrue)/eTrue // Abs;
    data = Transpose @ {shots, errs};

    ListLogLinearPlot[
        data,
        AxesLabel → {"shots", "error %"},
        PlotMarkers → Automatic, Joined→True,
        PlotLabel → If[label===None, circ, label]
    ]
]
```

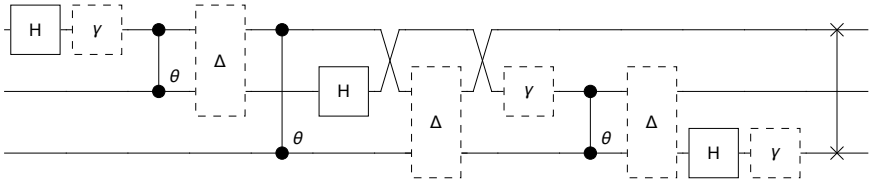
```
u = Circuit[
    H0 Damp0[.1] Deph1[.5] Depol2[.5] C0[X1] Deph1,2[.5] C2[S1] Depol0,1[.7] SWAP1,2];
DrawCircuit[u]
```



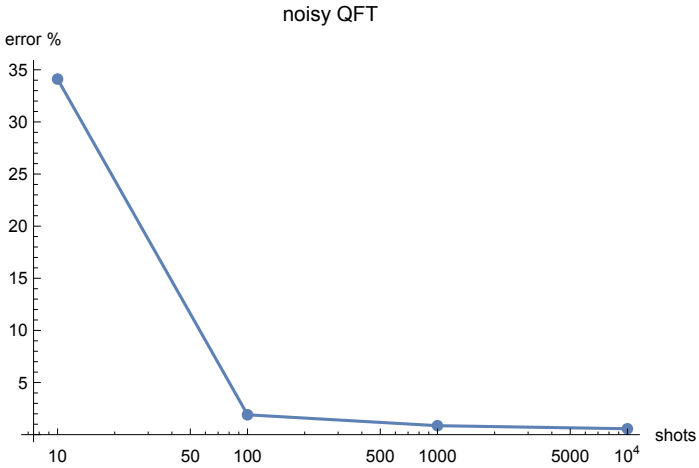

```
plotSampleErrors[u, samps, "circ"]
```



```
u = GetKnownCircuit["QFT", n];
u = u /. {g : _q_ :> {g, Damp_q[.5]}, g : _q1_,q2_[_] :> {g, Depol_q1,q2[.9]}};
u = Flatten @ u;
DrawCircuit[u]
```




```
plotSampleErrors[u, {10, 10^2, 10^3, 10^4}, "noisy QFT"]
```



Options

ShowProgress

```
u = Table[Depol_0,1[.5], 10^4];
h = GetRandomPauliString[n];
SampleExpecPauliString[ψi, u, h, 10^4, ShowProgress → True]
```

```
-0.908556
```

Working registers

```
tmp = CreateQuregs[n, 2];

Timing @ SampleExpecPauliString[ψi, Deph0,1[.2], h, All]
Timing @ SampleExpecPauliString[ψi, Deph0,1[.2], h, All, tmp]

{First@% < First @ %%, Last @ % === Last @ %%}
```

```
{0.006356, 8.50575}
{0.004842, 8.50575}
{True, True}
```

Warnings

```
SampleExpecPauliString[ψi, Deph0,1[.2], h, 100]
```

SampleExpecPauliString: As many or more samples were requested than there are unique decompositions of the circuit (of which there are 4). Proceeding instead with deterministic simulation of each decomposition in-turn. Hide this warning by setting the number of samples to All, or using Quiet[].

```
0.362668
```

Errors

```
SampleExpecPauliString[ψi, Deph0,1[.2], h, 1, CreateQuregs[2, 2]]
```

SampleExpecPauliString: The working quregs must have the same number of qubits as the initial qureg.

```
$Failed
```

```
SampleExpecPauliString[ψi, Deph0,1[.2], h, 1, CreateQuregs[n, 1]]
SampleExpecPauliString[ψi, Deph0,1[.2], h, 1, CreateQuregs[n, 3]]
```

SampleExpecPauliString: Invalid arguments. See ?SampleExpecPauliString

```
$Failed
```

SampleExpecPauliString: Invalid arguments. See ?SampleExpecPauliString

```
$Failed
```

```
SampleExpecPauliString[ψi, Deph0,1[.2], h, 0]
SampleExpecPauliString[ψi, Deph0,1[.2], h, -10]
SampleExpecPauliString[ψi, Deph0,1[.2], h, 1.5]
```

SampleExpecPauliString: The number of samples must be a positive integer.

```
$Failed
```

SampleExpecPauliString: The number of samples must be a positive integer.

```
$Failed
```

SampleExpecPauliString: Invalid arguments. See ?SampleExpecPauliString

```
$Failed
```

```
SampleExpecPauliString[invalidψ, Deph0,1[.2], h, 1]
SampleExpecPauliString[ψi, invalidCirc, h, 1]
SampleExpecPauliString[ψi, Deph0,1[.2], invalidPauli, 1]
SampleExpecPauliString[ψi, Deph0,1[.2], h, invalidShots]
```

SampleExpecPauliString: Invalid arguments. See ?SampleExpecPauliString

```
$Failed
```

SampleExpecPauliString: Invalid arguments. See ?SampleExpecPauliString

```
$Failed
```

SampleExpecPauliString: Invalid arguments. See ?SampleExpecPauliString

```
$Failed
```

SampleExpecPauliString: Invalid arguments. See ?SampleExpecPauliString

```
$Failed
```