Real-time simulation

This notebook introduces quantum simulation, and explores simulating the real-time dynamics of a spin-ring Ising system. We demonstrate classical analytic and numerical treatments, then study Trotterisation in the absence and presence of decoherence.

***Contents*:**

Tyson Jones, 2024
Department of Materials, University of Oxford
tyson.jones.input@gmail.com


```
Import["https://qtechtheory.org/questlink.m"];
CreateDownloadedQuESTEnv[];
```

Analytic

In quantum simulation, we are interested in studying the properties of a time-dependent state $\left|\psi(t)\right\rangle$ as it evolves according to the physics of some Hamiltonian $\hat{H}$. Consider this arbitrary 3-qubit Hamiltonian specified in the Pauli basis:

```
nQb = 3;
```

$$h = X_0\, Y_1 + 2\, Y_2\, Z_0 - 3\, Z_0\, Z_1\, Z_2;$$

```
hMatr = CalcPauliExpressionMatrix[h]
```

```
SparseArray[ Specified elements: 24  Dimensions: {8, 8} ]
```

We'll study the evolution of $\left|\psi(t)\right\rangle$ from initial state $\left|\psi(0)\right\rangle = \left|0\right\rangle$ initial state.

```
ψ0 = UnitVector[2^nQb, 1]
```

```
{1, 0, 0, 0, 0, 0, 0, 0}
```

One way to obtain the future states $\left|\psi(t)\right\rangle$ is to numerically solve the Schrödinger equation:

$$i \frac{d}{dt} \left|\psi(t)\right\rangle = \hat{H} \left|\psi(0)\right\rangle$$

```
NDSolve[{ⅈ ψ'[t] == hMatr . ψ[t], ψ[0] == ψ0}, ψ, {t, 0, 4}];
ψ = %[[1, 1, 2]]
```

```
InterpolatingFunction[ Domain: {{0., 4.}} Output dimensions: {8} ]
```

```
ψ[0] // Chop
```

```
{1., 0, 0, 0, 0, 0, 0, 0}
```

```
ψ[.3] // Chop
```

```
{0.444122 + 0.700796 ⅈ, 0, 0, 0.170249 + 0.216967 ⅈ, 0.483035, 0, 0, 0. - 0.0475127 ⅈ}
```

We could instead obtain an analytic expression for $\left|\psi(t)\right\rangle = \hat{U}(t)\left|\psi(0)\right\rangle$ by symbolically constructing the unitary time evolution operator $\hat{U}(t) = e^{-i t \hat{H}}$

```
u[t_] = MatrixExp[-ⅈ t CalcPauliExpressionMatrix[h]];
MatrixForm @ u[t]
```

$$
\begin{pmatrix}
\frac{1}{2}\mathrm{Cos}[2\sqrt{2}\,t] + \frac{1}{2}\mathrm{Cos}[2\sqrt{5}\,t] + i\left(\frac{\mathrm{Sin}[2\sqrt{2}\,t]}{2\sqrt{2}} + \frac{\mathrm{Sin}[2\sqrt{5}\,t]}{\sqrt{5}}\right) & 0 \\
0 & \frac{1}{2}\mathrm{Cos}[2\sqrt{2}\,t] + \frac{1}{2}\mathrm{Cos}[2\sqrt{5}\,t] \\
0 & i\left(-\frac{1}{2}\mathrm{Cos}[2\sqrt{2}\,t] + \frac{1}{2}\mathrm{Cos}[2\sqrt{\right. \\
i\left(\frac{1}{2}\mathrm{Cos}[2\sqrt{2}\,t] - \frac{1}{2}\mathrm{Cos}[2\sqrt{5}\,t]\right) - \frac{\mathrm{Sin}[2\sqrt{2}\,t]}{2\sqrt{2}} + \frac{\mathrm{Sin}[2\sqrt{5}\,t]}{\sqrt{5}} & 0 \\
\frac{\mathrm{Sin}[2\sqrt{2}\,t]}{2\sqrt{2}} + \frac{\mathrm{Sin}[2\sqrt{5}\,t]}{2\sqrt{5}} & 0 \\
0 & -\frac{\mathrm{Sin}[2\sqrt{2}\,t]}{2\sqrt{2}} - \\
0 & i\left(-\frac{\mathrm{Sin}[2\sqrt{2}\,t]}{2\sqrt{2}}\right. \\
i\left(-\frac{\mathrm{Sin}[2\sqrt{2}\,t]}{2\sqrt{2}} + \frac{\mathrm{Sin}[2\sqrt{5}\,t]}{2\sqrt{5}}\right) & 0
\end{pmatrix}
$$

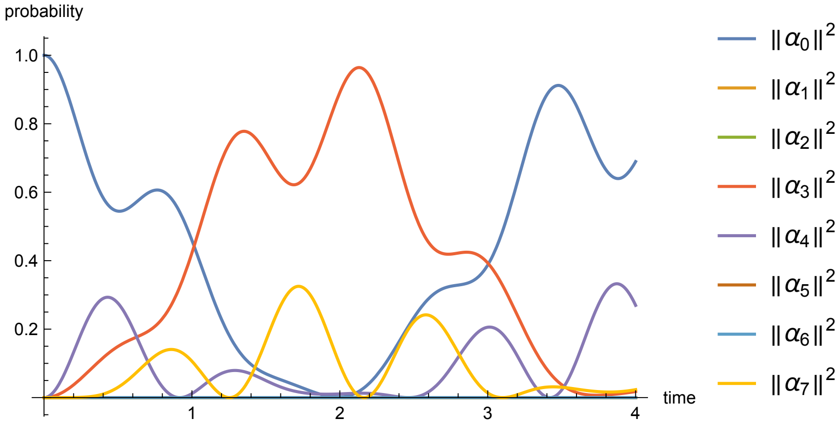
We can then obtain analytic expressions for the Z-basis amplitudes of $\left|\psi(t)\right\rangle$ as functions of t

```
Clear[ψ]
ψ[t] = u[t] . ψ0 // Simplify
```

$$
\left\{\frac{1}{20}\left(10\,\mathrm{Cos}[2\sqrt{2}\,t] + 10\,\mathrm{Cos}[2\sqrt{5}\,t] + 5\,i\sqrt{2}\,\mathrm{Sin}[2\sqrt{2}\,t] + 4\,i\sqrt{5}\,\mathrm{Sin}[2\sqrt{5}\,t]\right), 0, 0,\right.
$$
$$
\frac{1}{20}\left(10\,i\,\mathrm{Cos}[2\sqrt{2}\,t] - 10\,i\,\mathrm{Cos}[2\sqrt{5}\,t] - 5\sqrt{2}\,\mathrm{Sin}[2\sqrt{2}\,t] + 4\sqrt{5}\,\mathrm{Sin}[2\sqrt{5}\,t]\right),
$$
$$
\frac{1}{20}\left(5\sqrt{2}\,\mathrm{Sin}[2\sqrt{2}\,t] + 2\sqrt{5}\,\mathrm{Sin}[2\sqrt{5}\,t]\right), 0,
$$
$$
\left.0, -\frac{1}{20}\,i\left(5\sqrt{2}\,\mathrm{Sin}[2\sqrt{2}\,t] - 2\sqrt{5}\,\mathrm{Sin}[2\sqrt{5}\,t]\right)\right\}
$$

Here's how the probability of the basis states, with amplitudes α_i, evolve from $\delta_{i,0}$

```
probs = Abs[u[t] . ψ0]^2;

Plot[probs, {t, 0, 4},
    PlotRange → All,
    AxesLabel → {"time", "probability"},
    PlotLegends → Table[Norm[α_i]^2, {i, 0, 2^nQb - 1}]]
```

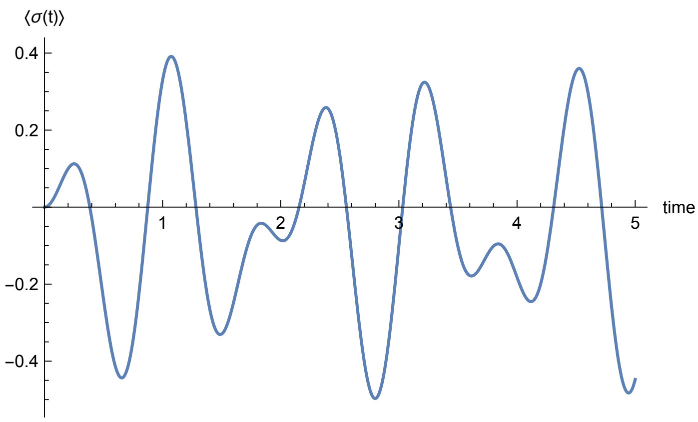


In quantum simulation, we are more often interested in the time evolution of the expectation value of some observable $\langle\sigma(t)\rangle = \left\langle\psi(t)\right| \hat{\sigma} \left|\psi(t)\right\rangle$. Consider this arbitrary Pauli operator:

```
σ = X_0 X_1 X_2;

v = Simplify[
     Conjugate[ψ[t]] . CalcPauliExpressionMatrix[σ] . ψ[t],
     t ≥ 0]
```

$$\frac{1}{20}\left(-1 + 5\,\mathrm{Cos}\left[4\sqrt{2}\,t\right] - 4\,\mathrm{Cos}\left[4\sqrt{5}\,t\right]\right)$$

```
Plot[v, {t, 0, 5}, AxesLabel → {"time", "⟨σ(t)⟩"}]
```



Numerical

Let’s switch to numerical simulation and choose a more interesting, physically-meaningful problem. We will simulate an Ising spin-ring, nominated for its potential utility in demonstrating quantum advantage, with (periodic) Hamiltonian:

$$\hat{H} = \sum_{i=0}^{nQb-1} \vec{\sigma}_i \cdot \vec{\sigma}_{i+1} + B \hat{Z}_i + d_i \hat{Z}_i$$

where $B = 4$ is the strength of a transverse magnetic field, and $d_i \in$ [-d, d]

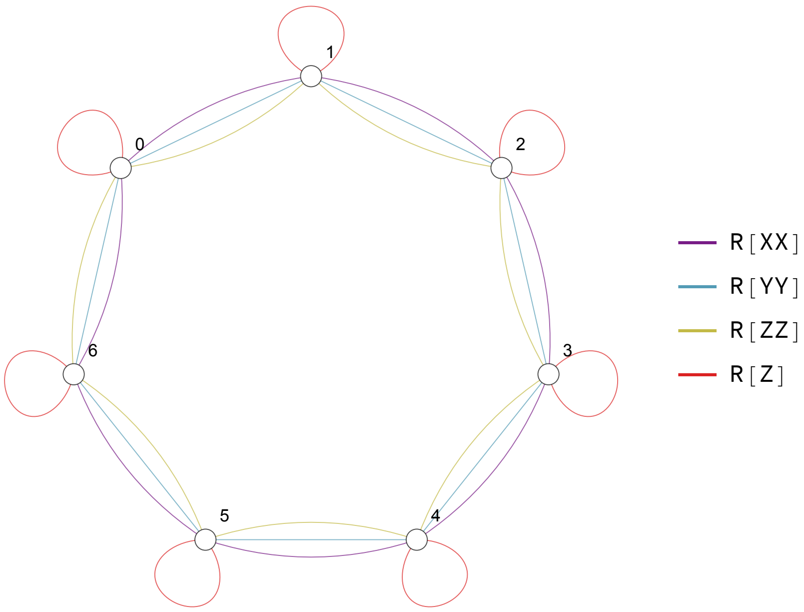
```
nQb = 7;
Clear[h]
```

```
h[d_] = Expand[
        Sum[(4 + d RandomReal[{-1, 1}]) Zi, {i, 0, nQb - 1}] +
        Sum[si sMod[i+1,nQb], {i, 0, nQb - 1}, {s, {X, Y, Z}}]]
```

$X_0 X_1 + X_1 X_2 + X_2 X_3 + X_3 X_4 + X_4 X_5 + X_0 X_6 + X_5 X_6 + Y_0 Y_1 + Y_1 Y_2 + Y_2 Y_3 + Y_3 Y_4 + Y_4 Y_5 + Y_0 Y_6 + Y_5 Y_6 + 4 Z_0 + 0.889235 d Z_0 + 4 Z_1 + 0.115662 d Z_1 + Z_0 Z_1 + 4 Z_2 + 0.15242 d Z_2 + Z_1 Z_2 + 4 Z_3 - 0.917439 d Z_3 + Z_2 Z_3 + 4 Z_4 - 0.606932 d Z_4 + Z_3 Z_4 + 4 Z_5 + 0.411064 d Z_5 + Z_4 Z_5 + 4 Z_6 + 0.0813288 d Z_6 + Z_0 Z_6 + Z_5 Z_6$

A quick way to confirm the ring topology of this system is to plot the connectivity of its Trotter circuit.

```
DrawCircuitTopology @ GetKnownCircuit["Trotter", h[1], 1, 1, t]
```



The uniformly random real scalars $d_i \in [-d, d]$ in $\hat{H}$ vary the effective magnetic field experienced by the spins. The scalar $d > 0$ is the strength of the *disorder* of the field.

```
CalcPauliStringMinEigVal @ h[1]
```

-21.6352

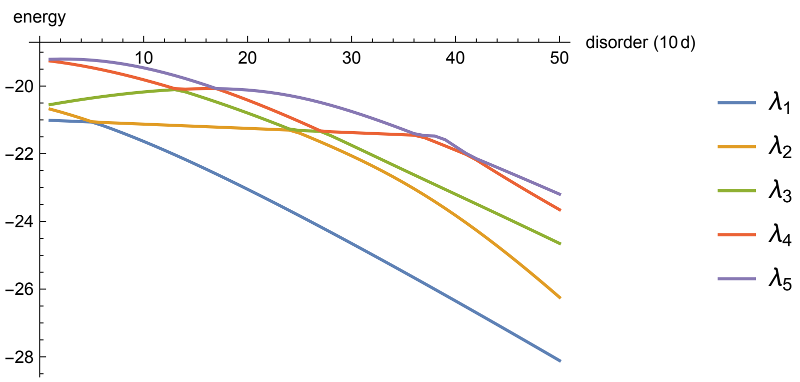
```
CalcPauliStringMinEigVal @ h[10]
```

```
-41.2231
```

This greatly affects the spectrum λ_j

```
vals = Transpose @ Table[
        - Eigenvalues[
            - CalcPauliExpressionMatrix @ h[d], 5,
            Method → {"Arnoldi", "Criteria" → "RealPart"}],
        {d, .1, 5, .1}];
```

```
ListLinePlot[
    vals,
    AxesLabel → {"disorder (10 d)", "energy"},
    PlotLegends → Table[λi, {i, 5}]]
```



To simulate time evolution under $\hat{H}$, we will make use of its Z-basis matrix representation. Here we use a Mathematica trick to *save* the matrix being computed for a given value of *d* so that repeated calls to hMatr[*d*] below don't repeat the computation.

```
Clear[hMatr]
hMatr[d_] := hMatr[d] = CalcPauliExpressionMatrix @ h[d]
```

```
First @ Timing @ hMatr[.1]
```

```
0.03099
```

```
First @ Timing @ hMatr[.1]
First @ Timing @ hMatr[.1]
First @ Timing @ hMatr[.1]
```

$8. \times 10^{-6}$

$6. \times 10^{-6}$

$5. \times 10^{-6}$

Let's consider an initial state whereby *half* of the spins are excited against the external field; this is the "Néel ordered state".

```
inψ = CreateQureg[nQb];
ApplyCircuit[inψ, Table[X_q, {q, 0, nQb - 1, 2}]];
```

```
GetQuregState[inψ, "ZBasisKets"]
```

|1010101⟩

We now *numerically* construct the time evolution operator in order to obtain future states.

```
trueψ = CreateQureg[nQb];

setTrueState[trueψ_, inψ_, hMatr_, t_] :=
    SetQuregMatrix[trueψ, MatrixExp[-i t hMatr] . GetQuregState[inψ]]
```

Time evolution under this Hamiltonian quickly excites the other spins:

```
setTrueState[trueψ, inψ, hMatr[.1], 0];
GetQuregState[trueψ, "ZBasisKets"]
```

|1010101⟩

```
setTrueState[trueψ, inψ, hMatr[.1], 10^-6];
GetQuregState[trueψ, "ZBasisKets"] // Chop
```

(0. − 2. × 10^-6 i) |0110101⟩ − (0. + 2. × 10^-6 i) |1001101⟩ −
 (0. + 2. × 10^-6 i) |1010011⟩ + (1. + 9.09068 × 10^-6 i) |1010101⟩ −
 (0. + 2. × 10^-6 i) |1010110⟩ − (0. + 2. × 10^-6 i) |1011001⟩ − (0. + 2. × 10^-6 i) |1100101⟩

```
setTrueState[trueψ, inψ, hMatr[.1], 1];
GetQuregState[trueψ, "ZBasisKets"][[;; 10]]
```

(−0.0233376 + 0.0143622 i) |0001111⟩ − (0.0870519 + 0.171909 i) |0010111⟩ +
 (0.0767504 − 0.0120878 i) |0011011⟩ − (0.0531524 + 0.103575 i) |0011101⟩ −
 (0.103286 − 0.0211168 i) |0011110⟩ − (0.120611 − 0.0299637 i) |0100111⟩ −
 (0.0736363 − 0.129366 i) |0101011⟩ − (0.197389 − 0.338313 i) |0101101⟩ +
 (0.167059 − 0.0165693 i) |0101110⟩ + (0.0198486 − 0.0990428 i) |0110011⟩

In this system, we are interested in the single-site magnetisation of the spins, $\langle Z_i \rangle$.

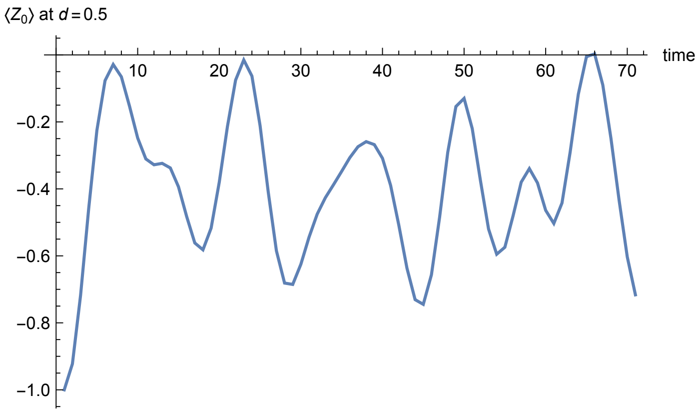
```
ϕ = CreateQureg[nQb];
CalcExpecPauliString[trueψ, Z_0, ϕ]
```

−0.163828

Let’s check how the magnetisation of the first qubit (which started *anti*-aligned with the transverse magnetic field) evolves in time, for a specific choice of disorder *d*.

```
pureData = Table[
        setTrueState[trueψ, inψ, hMatr[.5], t];
        CalcExpecPauliString[trueψ, Z0, ϕ],
        {t, 0, nQb, .1}];
```

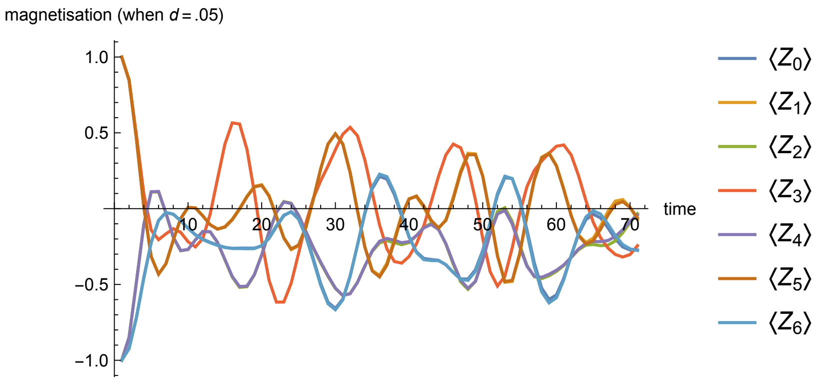
```
ListLinePlot[pureData, AxesLabel → {"time", "〈Z0〉 at d = 0.5"}]
```



Here's how the magnetisation of *all* spins evolve in time, when the field has small disorder.

```
data = Transpose@ Table[
        setTrueState[trueψ, inψ, hMatr[.01], t];
        CalcExpecPauliString[trueψ, Zq, ϕ],
        {t, 0, nQb, .1},
        {q, 0, nQb - 1}];
```

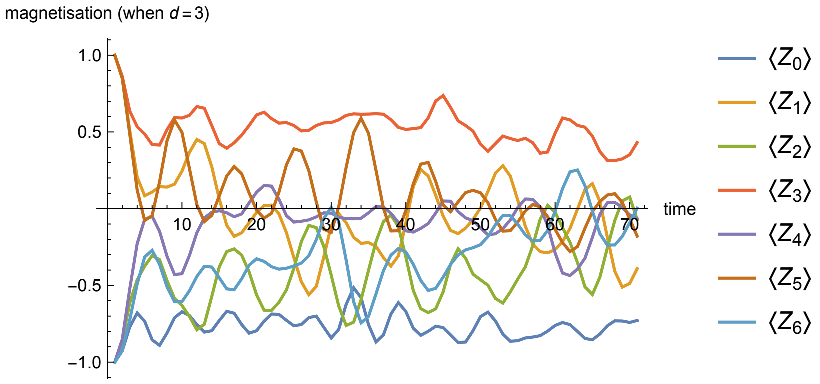
```
ListLinePlot[data,
     AxesLabel → {"time", "magnetisation (when d = .05)"},
     PlotLegends → Table[Row@{"〈", Zi, "〉"}, {i, 0, nQb - 1}]]
```



We see that the strong ±1 magnetisation of our initial state is quickly thermalized, and averages around zero. As reported here, the reason that this system is interesting is because as the disorder d of the transverse magnetic field is increased, the spins take significantly *longer* to thermalize. A strongly disordered external field sees the spins remain *localized*, retaining information about their original magnetisation. Here's the magnetisation in-time when the disorder is $d = 3$...

```
data = Transpose@ Table[
        setTrueState[trueψ, inψ, hMatr[5], t];
        CalcExpecPauliString[trueψ, Z_q, ϕ],
        {t, 0, nQb, .1},
        {q, 0, nQb - 1}];
```

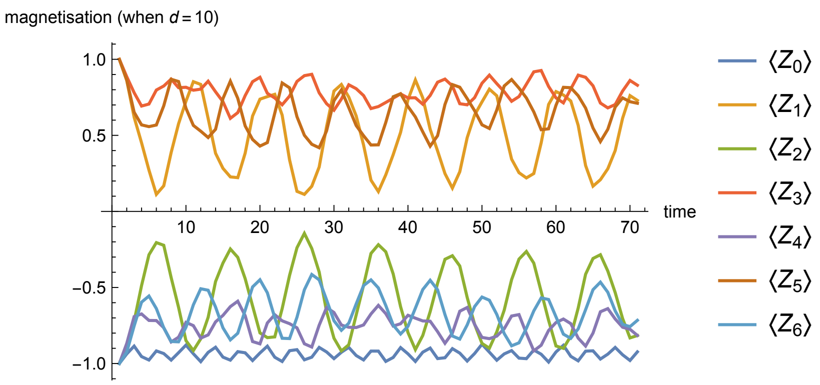
```
ListLinePlot[data,
     AxesLabel → {"time", "magnetisation (when d = 3)"},
     PlotLegends → Table[Row@{"⟨", Z_i, "⟩"}, {i, 0, nQb - 1}]]
```



and when $d = 10$...

```
data = Transpose@ Table[
        setTrueState[trueψ, inψ, hMatr[10], t];
        CalcExpecPauliString[trueψ, Z_q, ϕ],
        {t, 0, nQb, .1},
        {q, 0, nQb - 1}];
```

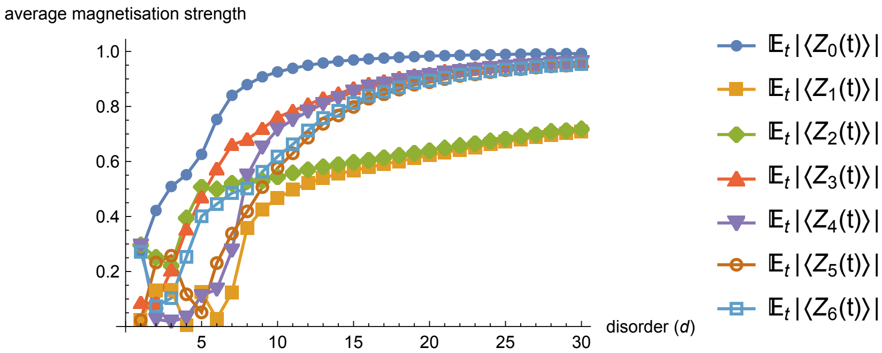
```
ListLinePlot[data,
     AxesLabel → {"time", "magnetisation (when d = 10)"},
     PlotLegends → Table[Row@{"⟨", Z_i, "⟩"}, {i, 0, nQb - 1}]]
```



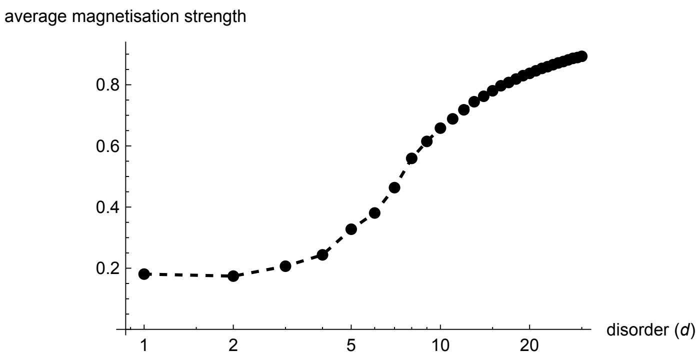
There is in fact a *phase transition* in the magnetisation occurring over d.

```
data = Transpose @ Table[
    Abs /@ Mean /@ Transpose @ Table[
        setTrueState[trueψ, inψ, hMatr[d], t];
        CalcExpecPauliString[trueψ, Zq, ϕ],
        {t, 0, nQb / 2, .1},
        {q, 0, nQb - 1}],
    {d, .01, 30, 1}];
```

```
ListLinePlot[data,
    AxesLabel → {"disorder (d)", "average magnetisation strength"},
    PlotLegends → Table[Row@{"𝔼t | ⟨", Zi, "(t)⟩ |"}, {i, 0, nQb - 1}],
    PlotMarkers → Automatic
]
```




```
ListLogLinearPlot[
    Mean @ data,
    AxesLabel → {"disorder (d)", "average magnetisation strength"},
    Joined → True, PlotMarkers → Automatic,
    PlotStyle → Directive[Black, Dashed]
]
```



Trotterisation

Pure

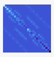
Let’s pretend the previous section’s spin-ring Hamiltonian was sadly *too big* to process by the classical numerics above, though it remains tractable to describe as a Pauli string:

```
h1 = h[1]
```

$$X_0 X_1 + X_1 X_2 + X_2 X_3 + X_3 X_4 + X_4 X_5 + X_0 X_6 + X_5 X_6 + Y_0 Y_1 + Y_1 Y_2 + Y_2 Y_3 + Y_3 Y_4 + Y_4 Y_5 + Y_0 Y_6 + Y_5 Y_6 + 4.88923\, Z_0 + 4.11566\, Z_1 + Z_0 Z_1 + 4.15242\, Z_2 + Z_1 Z_2 + 3.08256\, Z_3 + Z_2 Z_3 + 3.39307\, Z_4 + Z_3 Z_4 + 4.41106\, Z_5 + Z_4 Z_5 + 4.08133\, Z_6 + Z_0 Z_6 + Z_5 Z_6$$

Suppose the Z-basis representation of $\hat{H}$ is classically computationally intractable, so we could *not* evaluate this:

```
h1Matr = CalcPauliExpressionMatrix[h1]
```

SparseArray[ Specified elements: 1024 Dimensions: {128, 128}]

Imagine that we fortunately we have a quantum computer at our disposal, with which to perform quantum simulation! The canonical method of quantumly simulating the real-time unitary dynamics of a system is to evaluate the circuit produced by *Trotterising* its unitary time evolution operator:

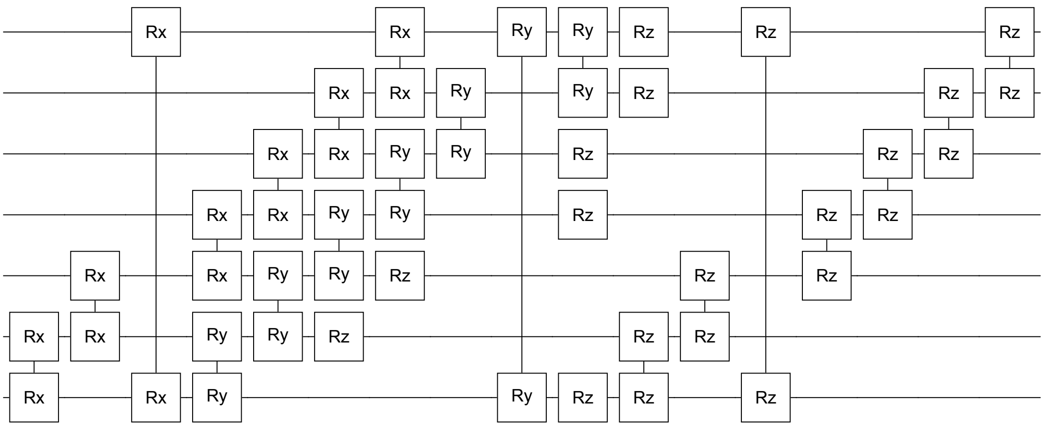
$$e^{-i\, t\, \hat{H}} \approx \prod_i \hat{U}_i[t]$$

```
Clear[u]
u[t_] = GetKnownCircuit["Trotter", h1, 1, 1, t]
```

$\{R[2t, X_0 X_1], R[2t, X_1 X_2], R[2t, X_2 X_3], R[2t, X_3 X_4], R[2t, X_4 X_5], R[2t, X_0 X_6], R[2t, X_5 X_6], R[2t, Y_0 Y_1], R[2t, Y_1 Y_2], R[2t, Y_2 Y_3], R[2t, Y_3 Y_4], R[2t, Y_4 Y_5], R[2t, Y_0 Y_6], R[2t, Y_5 Y_6], R[9.77847\, t, Z_0], R[8.23132\, t, Z_1], R[2t, Z_0 Z_1], R[8.30484\, t, Z_2], R[2t, Z_1 Z_2], R[6.16512\, t, Z_3], R[2t, Z_2 Z_3], R[6.78614\, t, Z_4], R[2t, Z_3 Z_4], R[8.82213\, t, Z_5], R[2t, Z_4 Z_5], R[8.16266\, t, Z_6], R[2t, Z_0 Z_6], R[2t, Z_5 Z_6]\}$

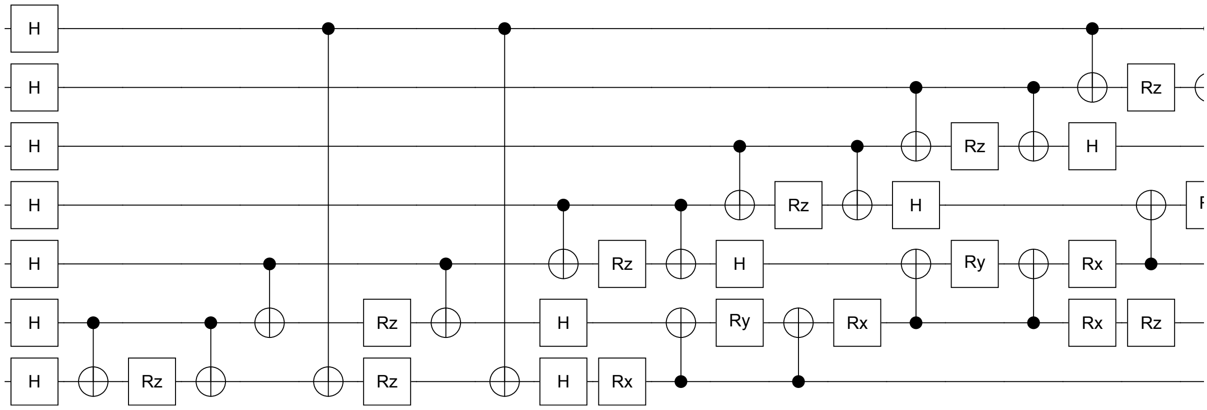
This is a circuit with gate parameters dependent upon the coefficients of our Hamiltonian, and the target simulation time *t*.

```
DrawCircuit @ u[t]
```

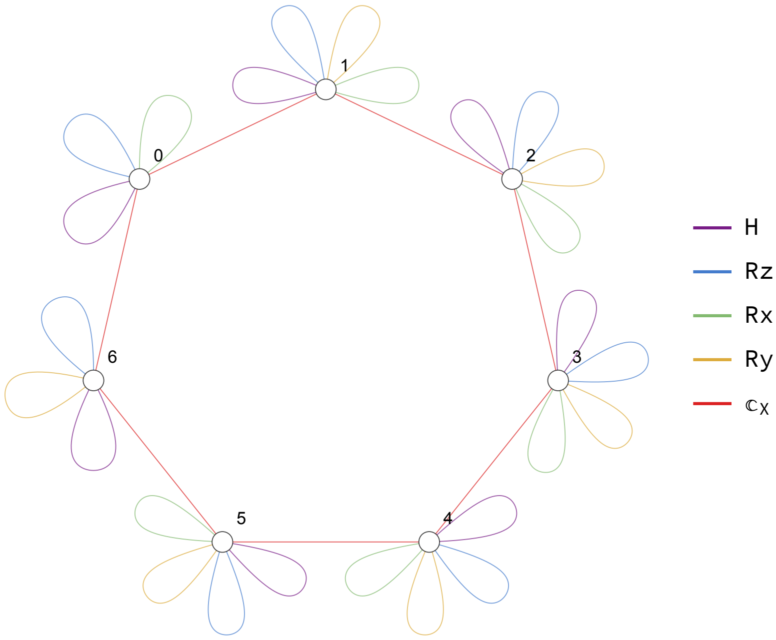


If necessary, we could recompile this circuit into the native operations of our hardware...

```
v = SimplifyCircuit @ RecompileCircuit[u[t], "SingleQubitAndCNOT"];
DrawCircuit[v]
```




```
DrawCircuitTopology @ v
```



but for convenience, let's assume our hardware can perform the original two-qubit Pauli gadgets. The Trotter circuit $\hat{U}$, applied to the initial state $|\psi(0)\rangle$, produces a direct approximation to the state $|\psi(t)\rangle = e^{-i t \hat{H}} |\psi(0)\rangle$

```
ψ = CreateQureg[nQb];
CloneQureg[ψ, inψ];
GetQuregState[ψ, "ZBasisKets"]
```

$|1010101\rangle$

```
τ = 0.1;
ApplyCircuit[ψ, u[τ]];
setTrueState[trueψ, inψ, h1Matr, τ];
```

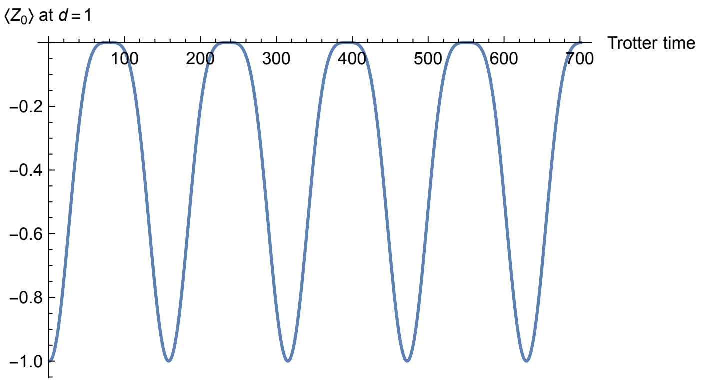
```
CalcFidelity[ψ, trueψ]
```

```
0.992173
```

An experimentalist can ergo apply circuit $\hat{U}(t)$, with rotations informed by the desired simulation time t, and thereafter measure their observable of interest, like the spin-ring magnetisation.

```
data = Table[
        CloneQureg[ψ, inψ];
        ApplyCircuit[ψ, u[t]];
        CalcExpecPauliString[ψ, Z0, ϕ],
        {t, 0, nQb, .01}];
```

```
ListLinePlot[data, AxesLabel → {"Trotter time", "⟨Z0⟩ at d = 1"}]
```



This doesn't look how we expected - it is suspiciously periodic (as the Trotter circuit *is* as a function of t), whereas we earlier witnessed thermalisation. Indeed the fidelity is imperfect - Trotterisation can only *approximate* the evolution when the Hamiltonian contains non-commuting terms.

```
τ = 0.3;
ApplyCircuit[CloneQureg[ψ, inψ], u[τ]];
setTrueState[trueψ, inψ, h1Matr, τ];
```

```
CalcFidelity[ψ, trueψ]
```

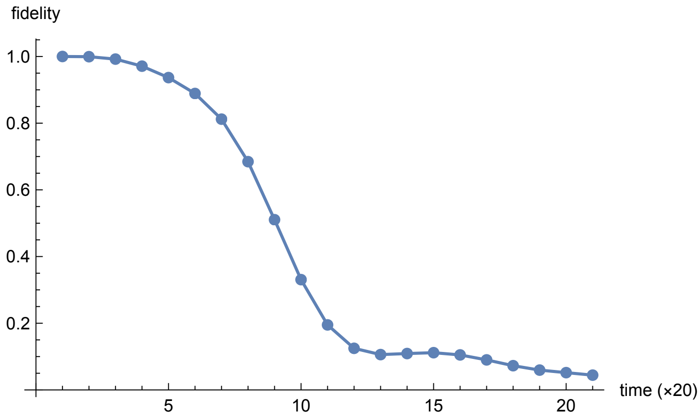
```
0.812077
```

The fidelity $\left| \langle \psi(0) | e^{i t \hat{H}} \hat{U}(t) | \psi(0) \rangle \right|^2$ drops quickly with increasing t.

```
Bra[ψ]
```

$\langle 32 |$

```
fid = Table[
        ApplyCircuit[CloneQureg[ψ, inψ], u[t]];
        setTrueState[trueψ, inψ, h1Matr, t];
        CalcFidelity[ψ, trueψ],
        {t, 0, 1, .05}];

ListLinePlot[fid,
      AxesLabel → {"time (×20)", "fidelity"},
      PlotMarkers → Automatic
]
```



We can improve the fidelity by using more Trotter *repetitions*, or using a *higher order* method.

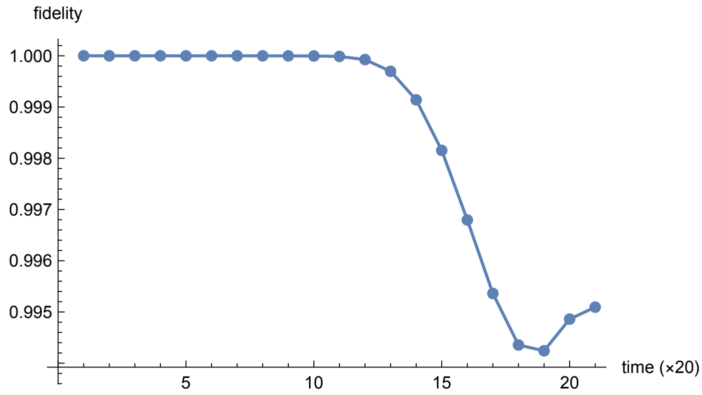
```
order = 4;
reps = 3;
u[t_] = GetKnownCircuit["Trotter", h1, order, reps, t]
```

$\left\{R\left[\frac{t}{3\left(4-2^{2/3}\right)}, X_0 X_1\right], R\left[\frac{t}{3\left(4-2^{2/3}\right)}, X_1 X_2\right], R\left[\frac{t}{3\left(4-2^{2/3}\right)}, X_2 X_3\right], R\left[\frac{t}{3\left(4-2^{2/3}\right)}, X_3 X_4\right],\right.$
$\left.\ldots 818 \ldots, R\left[\frac{t}{3\left(4-2^{2/3}\right)}, X_2 X_3\right], R\left[\frac{t}{3\left(4-2^{2/3}\right)}, X_1 X_2\right], R\left[\frac{t}{3\left(4-2^{2/3}\right)}, X_0 X_1\right]\right\}$

large output | show less | show more | show all | set size limit...

```
fid = Table[
        ApplyCircuit[CloneQureg[ψ, inψ], u[t]];
        setTrueState[trueψ, inψ, h1Matr, t];
        CalcFidelity[ψ, trueψ],
        {t, 0, 1, .05}];
```

```
ListLinePlot[fid,
     AxesLabel → {"time (×20)", "fidelity"},
     PlotMarkers → Automatic
]
```



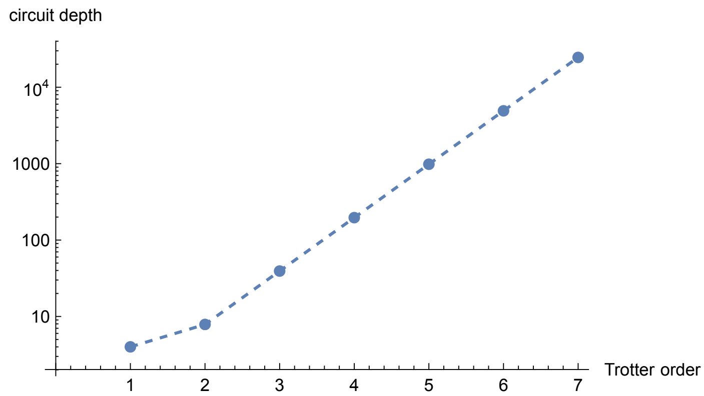
Of course, this means increasing the number of gates in the circuit.

```
Length @ GetKnownCircuit["Trotter", h1, 4, 1, t]
```

```
275
```

```
cost = Table[
        Length @ GetKnownCircuit["Trotter", h1, order, 1, t] / nQb,
        {order, {1, 2, 4, 6, 8, 10, 12}}];
```

```
ListLogPlot[cost,
     AxesLabel → {"Trotter order", "circuit depth"},
     Joined → True, PlotMarkers → Automatic,
     PlotStyle → Directive[Dashed]
]
```

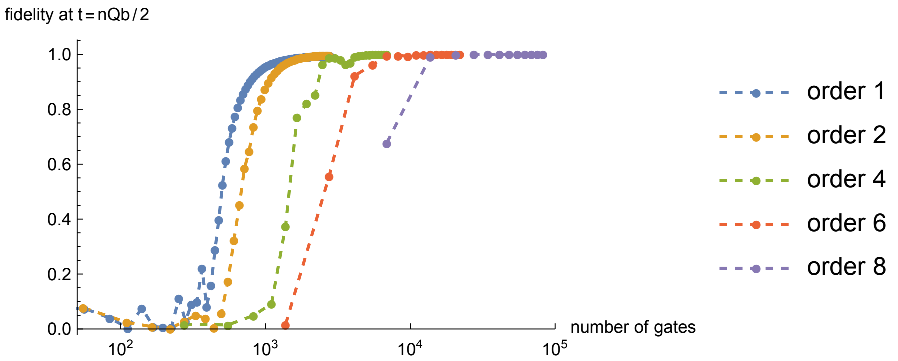


Let’s simulate to fixed time $t = nQb/2$, and record the costs and performance of using Trotter circuits of different order and repetitions.

```
τ = nQb / 2;
setTrueState[trueψ, inψ, h1Matr, τ];

data = Table[
        u = GetKnownCircuit["Trotter", h1, order, reps, N@τ];
        ApplyCircuit[CloneQureg[ψ, inψ], u];
        {Length[u], CalcFidelity[ψ, trueψ]},
        {order, {1, 2, 4, 6, 8}},
        {reps, 1, Floor[100 / order]}];
```

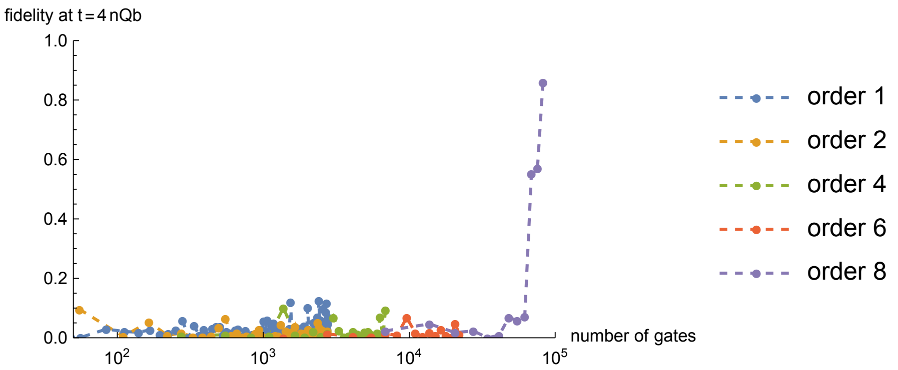
We can see that higher-order Trotter quickly becomes *expensive*:

```
ListLogLinearPlot[data,
    AxesLabel → {"number of gates", "fidelity at t = nQb / 2"},
    Joined → True, PlotRange → {{.5 × 10^2, 10^5}, All},
    PlotStyle → Dashed, PlotMarkers → {"●", 7},
    Ticks → {Table[{10^i, "10"^i}, {i, 1, 5}], Automatic},
    PlotLegends → Table["order " <> ToString[i], {i, {1, 2, 4, 6, 8}}]
]
```



yet it is our only hope if we wish to simulate *far* into the future!

```
τ = 4 nQb;
setTrueState[trueψ, inψ, h1Matr, τ];

data = Table[
        u = GetKnownCircuit["Trotter", h1, order, reps, N@τ];
        ApplyCircuit[CloneQureg[ψ, inψ], u];
        {Length[u], CalcFidelity[ψ, trueψ]},
        {order, {1, 2, 4, 6, 8}},
        {reps, 1, Floor[100 / order]}];

ListLogLinearPlot[data,
      AxesLabel → {"number of gates", "fidelity at t = 4 nQb"},
      Joined → True, PlotRange → {{.5 × 10^2, 10^5}, {0, 1}},
      PlotStyle → Dashed, PlotMarkers → {"●", 7},
      Ticks → {Table[{10^i, "10"^i}, {i, 1, 5}], Automatic},
      PlotLegends → Table["order " <> ToString[i], {i, {1, 2, 4, 6, 8}}]
]
```



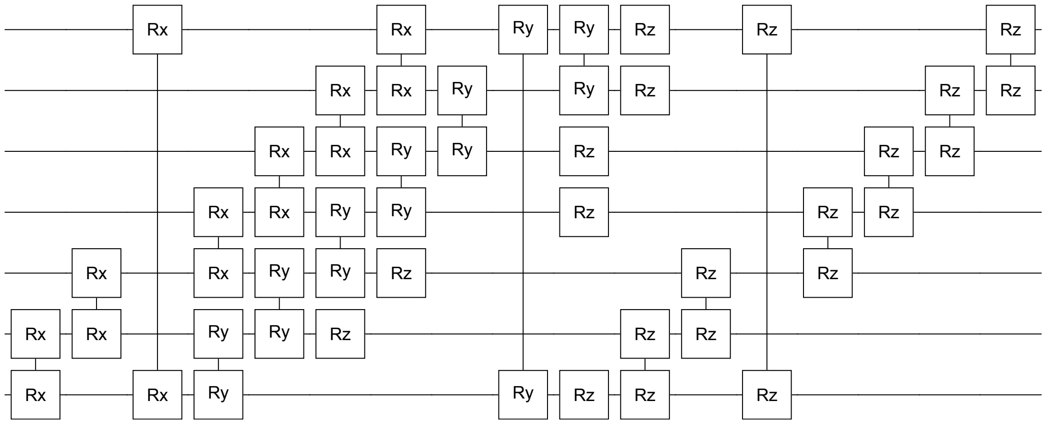
Noisy

There is yet another problem - what happens if our quantum hardware is *imperfect* and susceptible to decoherence? Let's now assume that parameter-dependent dephasing noise of probability $\xi\,|\theta|$ follows every Rz[θ] gate, and fixed two-qubit depolarising noise of strength ξ follows every two-qubit Pauli gadget.

```
noisify[u_, ξ_] := u /. {
        g : R[θ_, Z_t_] :→ Sequence[g, Deph_t[Abs[θ] ξ]],
        g : R[θ_, Verbatim[Times][_t1_, _t2_]] :→ Sequence[g, Depol_t1,t2[ξ]]}
```

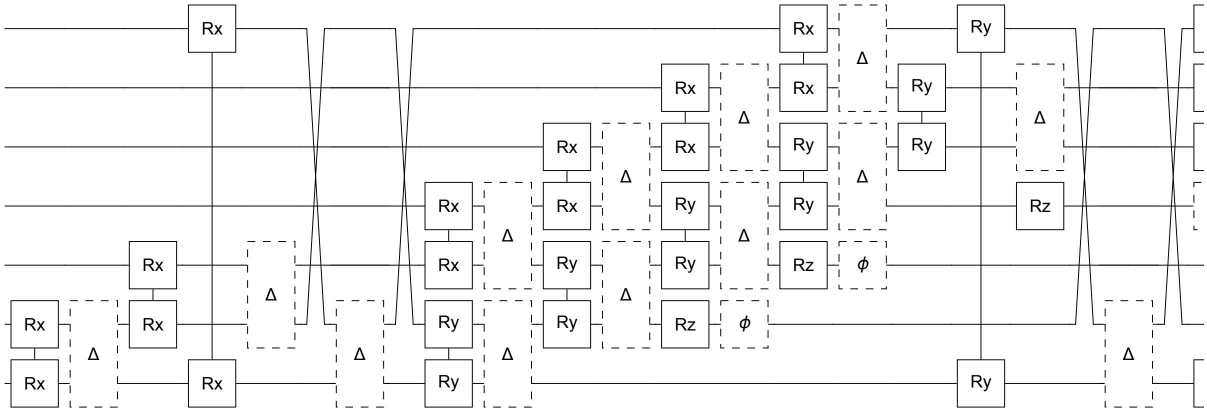
Attempting to perform the first-order single-repetition unitary Trotter circuit...

```
τ = 0.1;
u = GetKnownCircuit["Trotter", h1, 1, 1, τ];
DrawCircuit[u]
```



would instead invoke the channel:

```
DrawCircuit @ noisify[u, ξ]
```



Simulating this channel will require we switch our quantum registers to be *density matrices*.

```
ρ = CreateDensityQureg[nQb];
InitPureState[ρ, inψ];
ApplyCircuit[ρ, noisify[u, 10^-3]];
CalcPurity[ρ]
0.953016
```

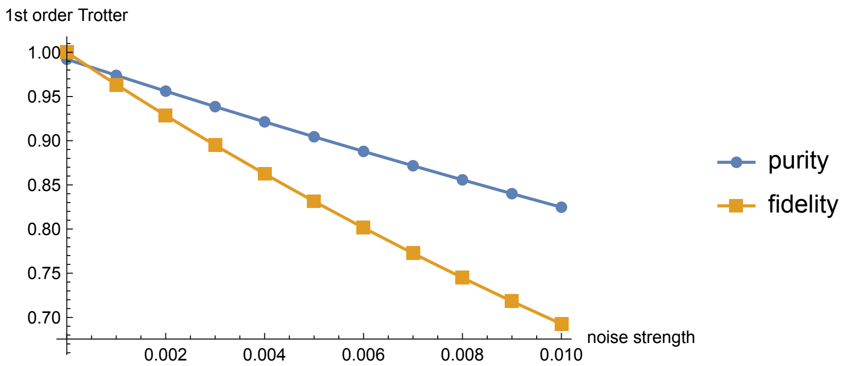
The decoherence (*physical error*) worsens our fidelity, compounding the existing inaccuracy of our Trotter truncation (*algorithmic error*)

```
setTrueState[trueψ, inψ, h1Matr, τ];

InitPureState[ρ, inψ];
ApplyCircuit[ρ, noisify[u, 10^-2]];
CalcFidelity[ρ, trueψ]
```

```
0.824697
```

Here’s first-order single-repetition Trotter simulation succumbing to increasing decoherence.

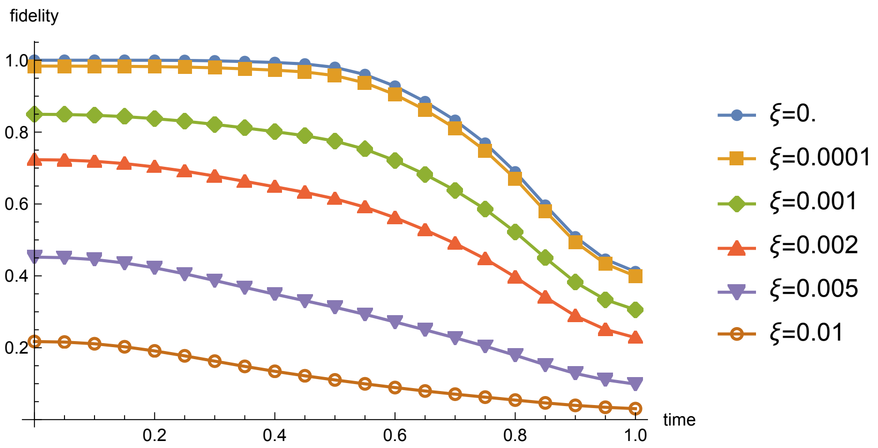
```
noise = Range[0, 10^-2, 10^-3];
data = Transpose @ Table[
        InitPureState[ρ, inψ];
        ApplyCircuit[ρ, noisify[u, ξ]];
        {CalcFidelity[ρ, trueψ], CalcPurity[ρ]},
        {ξ, noise}];

ListLinePlot[
     Transpose[{noise, #}] & /@ data,
     AxesLabel → {"noise strength", "1st order Trotter"},
     PlotLegends → {"purity", "fidelity"},
     PlotMarkers → Automatic
]
```



While the algorithmic Trotter error worsens for increasing *t*, the physical error of decoherence worsens the fidelity at *all times*. Even a measly *t=1* simulation is quickly ruined!

```
Clear[u];
u[t_] = GetKnownCircuit["Trotter", h1, 4, 1, t];
ch[t_, ξ_] = noisify[u[t], ξ];

time = Range[0, 1, 0.05];
noise = {0, 10^-4, 10^-3, 2 × 10^-3, 5 × 10^-3, 10^-2};
fid = Transpose @ Table[
        setTrueState[trueψ, inψ, h1Matr, t];
        ApplyCircuit[InitPureState[ρ, inψ], ch[t, ξ]];
        CalcFidelity[ρ, trueψ],
        {t, time},
        {ξ, noise}];
```

```
ListLinePlot[
    Transpose[{time, #}] & /@ fid,
    AxesLabel → {"time", "fidelity"},
    PlotLegends → Table["ξ=" <> ToString[N@ξ], {ξ, noise}],
    PlotMarkers → Automatic
]
```

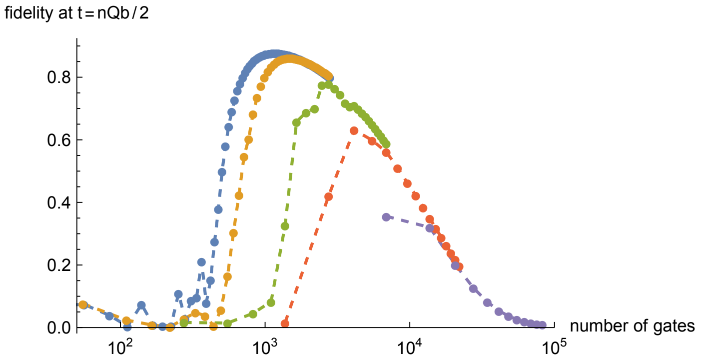


Let's repeat our earlier resource-aware simulations of time $t = nQb/2$ using increasing Trotter orders and repetitions, but this time incorporating a modest physical error rate of $\xi = 10^{-4}$.

```
τ = nQb / 2;
ξ = 10^-4;
setTrueState[trueψ, inψ, h1Matr, τ];

data = Table[
     u = GetKnownCircuit["Trotter", h1, order, reps, N@τ];
     ApplyCircuit[InitPureState[ρ, inψ], noisify[u, ξ]];
     {Length[u], CalcFidelity[ρ, trueψ]},
     {order, {1, 2, 4, 6, 8}},
     {reps, 1, Floor[100 / order]}];
```

This reveals increasing the Trotter repetitions and order can actually *worsen* performance, because the additional gates introduce more opportunities for physical error:

```
ListLogLinearPlot[data,
    AxesLabel → {"number of gates", "fidelity at t = nQb / 2"},
    Joined → True, PlotRange → {{.5 × 10^2, 10^5}, All},
    PlotStyle → Dashed, PlotMarkers → {"●", 7},
    Ticks → {Table[{10^i, "10"^i}, {i, 1, 5}], Automatic},
    PlotLegends → Table["order " <> ToString[i], {i, {1, 2, 4, 6, 8}}]
]
```



This is why Trotterisation is believed to be incompatible with near-future noisy quantum hardware. It prescribes deep circuits leveraging precise interference effects, which are easily damaged by noise and the imperfections of near-future quantum computers.

How would the experimentalist fair attempting to use this hardware to study the dynamics of magnetisation in our spin-ring system?

```
μ = CreateDensityQureg[nQb];
```

```
data = Table[
    u = GetKnownCircuit["Trotter", h1, 4, 1, t];
    ApplyCircuit[InitPureState[ρ, inψ], noisify[u, 10^-4]];
    CalcExpecPauliString[ρ, Z0, μ],
    {t, 0, nQb, .1}];
```

Well at least it more closely resembles thermalization! ¯_(ツ)_/¯

```
ListLinePlot[
    {data, pureData},
    AxesLabel → {"time (×10)", "⟨Z₀⟩ at d = 1"},
    PlotMarkers → Automatic,
    PlotLegends → {"ξ=10^-4", "ξ=0"}]
```