This is the command to get the latest release of QuESTlink:

```
(* Import["https://qtechtheory.org/questlink.m"]; *)
```

but this notebook will instead use a developmental version of QuESTlink (for nicer **DrawCircuit** rendering)

```
Import[
 "https://raw.githubusercontent.com/QTechTheory/QuESTlink/hardware-profiles/
   Link/QuESTlink.m"]
```

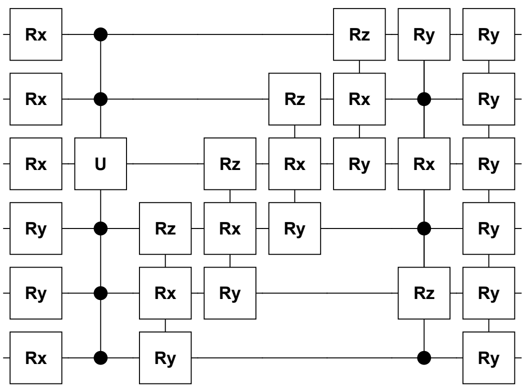
The next command downloads the pre-compiled single-thread simulator. For significantly faster multithreaded and GPU simulation, recompile QuESTlink (see guide here) and use **CreateLocalQuESTEnv[]**

```
CreateDownloadedQuESTEnv[];
```

Preparing an ansatz

Here's how we could hard-code a 6 qubit, 12 parameter ansatz. The parameters are labelled θ[i] for convenience, but they could be any symbols we want (θi, θ_i, etc)

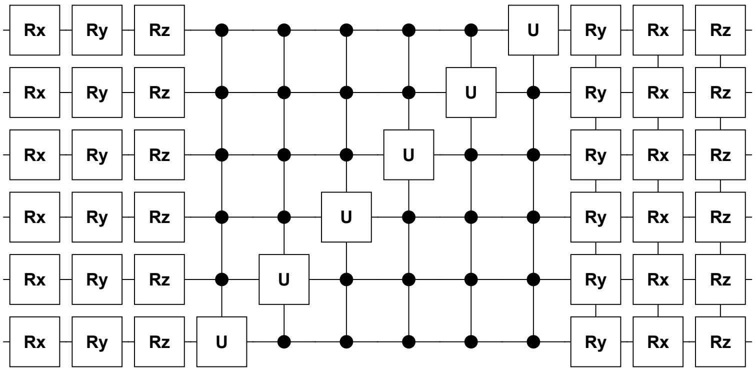
```
Circuit[
      Rx_0[θ[1]] Ry_1[θ[2]] Ry_2[θ[3]] Rx_3[θ[4]] Rx_4[θ[5]] Rx_5[θ[6]] ×
       C_{0,1,2,4,5}[U_3[( Exp[i θ[7]]          0
                                 0        Exp[-i θ[7]] )]] ×
       R[θ[7], Y_0 X_1 Z_2] R[θ[8], Y_1 X_2 Z_3] R[θ[9], Y_2 X_3 Z_4] R[θ[10], Y_3 X_4 Z_5] ×
       C_{0,2,4}[R[θ[11], Z_1 X_3 Y_5]] ×
       R[θ[12], Y_0 Y_1 Y_2 Y_3 Y_4 Y_5]];
DrawCircuit[%]
```



More often, we wish to generate an ansatz programmatically.

```
nQ = 6;
u = Block[{i = 1}, Flatten @ {
      Table[ g_(q-1)[θ[i++]], {q, nQ}, {g, {Rx, Ry, Rz}}],
      Table[C_(DeleteCases[Range[0,nQ-1],q]) [U_q[( Exp[i θ[i++]]  0
                                                   0              1 ) ]], {q, 0, nQ - 1}],
      Table[R[θ[i++], Product[σ_(q-1), {q, nQ}]],  {σ, {Y, X, Z}}]}];
nθ = Length[u];
```

```
DrawCircuit[u]
```



Since the parameters symbolic, we could study the ansatz operator analytically. Here’s its first element, when every second parameter has value π.

```
u /. θ[_?EvenQ] → π;
CalcCircuitMatrix[%]  // First // First
```

$$-i\, e^{\frac{1}{2} i \theta[3]+\frac{1}{2} i \theta[9]+\frac{1}{2} i \theta[15]-\frac{1}{2} i \theta[27]} \operatorname{Cos}\left[\frac{\theta[1]}{2}\right] \operatorname{Cos}\left[\frac{\theta[5]}{2}\right] \operatorname{Cos}\left[\frac{\theta[7]}{2}\right] \operatorname{Cos}\left[\frac{\theta[11]}{2}\right] \operatorname{Cos}\left[\frac{\theta[13]}{2}\right] \operatorname{Cos}\left[\frac{\theta[17]}{2}\right] \operatorname{Cos}\left[\frac{\theta[25]}{2}\right] - i\, e^{-\frac{1}{2} i \theta[3]-\frac{1}{2} i \theta[9]-\frac{1}{2} i \theta[15]-\frac{1}{2} i \theta[27]} \operatorname{Sin}\left[\frac{\theta[1]}{2}\right] \operatorname{Sin}\left[\frac{\theta[5]}{2}\right] \operatorname{Sin}\left[\frac{\theta[7]}{2}\right] \operatorname{Sin}\left[\frac{\theta[11]}{2}\right] \operatorname{Sin}\left[\frac{\theta[13]}{2}\right] \operatorname{Sin}\left[\frac{\theta[17]}{2}\right] \operatorname{Sin}\left[\frac{\theta[25]}{2}\right]$$

Preparing a Hamiltonian

We could hard-code our own Hamiltonian...

```
h = .1 X_0 Y_1 Z_2 + .3 Y_0 X_2 - .4 Z_0 Z_1 Z_2;
```

```
CalcPauliSumMatrix[h] // Chop // MatrixForm
```

$$\begin{pmatrix} -0.4 & 0 & 0 & 0.-0.1 i & 0 & 0.-0.3 i & 0 & 0 \\ 0 & 0.4 & 0.-0.1 i & 0 & 0.+0.3 i & 0 & 0 & 0 \\ 0 & 0.+0.1 i & 0.4 & 0 & 0 & 0 & 0 & 0.-0.3 i \\ 0.+0.1 i & 0 & 0 & -0.4 & 0 & 0 & 0.+0.3 i & 0 \\ 0 & 0.-0.3 i & 0 & 0 & 0.4 & 0 & 0 & 0.+0.1 i \\ 0.+0.3 i & 0 & 0 & 0 & 0 & -0.4 & 0.+0.1 i & 0 \\ 0 & 0 & 0 & 0.-0.3 i & 0 & 0.-0.1 i & -0.4 & 0 \\ 0 & 0 & 0.+0.3 i & 0 & 0.-0.1 i & 0 & 0 & 0.4 \end{pmatrix}$$

but more likely we'll want to produce one from a file of coefficients and Pauli strings. Here we download a 369-term 6-qubit Lithium Hydride Hamiltonian

```
h = GetPauliSumFromCoeffs[
   "https://questlink.qtechtheory.org/demo_hamiltonian.txt"];
```

```
Length[h]
h[[ ;; 30]]
```

369

$$-6.52209 - 0.00168947 X_0 + 0.000335609 X_1 + 0.00233908 X_0 X_1 - 0.00518865 X_2 - 2.32678 \times 10^{-6} X_0 X_2 - 0.00238276 X_1 X_2 - 0.000333484 X_0 X_1 X_2 + 0.0561302 X_3 + 0.0000211588 X_0 X_3 + 0.0000198838 X_0 X_1 X_3 - 0.000133652 X_2 X_3 - 0.0000311241 X_1 X_2 X_3 + 0.000547046 X_4 + 0.00165752 X_3 X_4 - 0.00600013 X_0 X_3 X_4 - 0.00812442 X_2 X_3 X_4 + 0.000447423 X_5 - 0.0000517994 X_0 X_5 + 0.00019084 X_0 X_1 X_5 - 0.0000777401 X_2 X_5 - 0.0001908 X_1 X_2 X_5 - 0.00630859 X_3 X_5 + 0.0661688 X_4 X_5 + 0.0000169999 X_0 X_4 X_5 + 0.00634842 X_0 X_1 X_4 X_5 - 0.000177071 X_2 X_4 X_5 - 0.00636179 X_1 X_2 X_4 X_5 + 0.00562722 X_3 X_4 X_5 - 0.000346402 Y_0 Y_1$$

We can determine its ground-state exactly, which we'll later compare to that which our variational algorithm estimates

```
gs = Min @ Eigenvalues @ CalcPauliSumMatrix[h]
```

-7.88074

Preparing quantum states

To perform numerical simulation of our ansatz , we need to create quantum registers, which are stored in the backend QuEST process.

```
ψ = CreateQureg[nQ];
InitPlusState[ψ];
```

```
GetQuregMatrix[ψ] // Chop
```

{0.125, 0.125}

To apply our ansatz circuit to a state, we must replace the parameters with numerical values

```
ApplyCircuit[u /. θ[_] :> RandomReal[], ψ];

GetQuregMatrix[ψ]
```

```
{0.0224731 + 0.111599 i, 0.00523535 - 0.0521917 i,
 -0.0111565 - 0.0767799 i, -0.00378012 + 0.00136288 i, 0.00123122 - 0.0731973 i,
 -0.00457615 + 0.00651081 i, 0.00978255 - 0.00067529 i,
 -0.0267522 - 0.0498707 i, -0.0235968 - 0.0586924 i, -0.0191137 - 0.00769477 i,
 -0.00438554 - 0.0141594 i, -0.00728107 - 0.0635589 i, -0.00530986 - 0.0091749 i,
 -0.0142068 - 0.0662965 i, -0.0103575 - 0.0461789 i, -0.00615036 - 0.134759 i,
 0.0153867 - 0.0281887 i, -0.0302696 + 0.0147383 i, -0.0151482 + 0.00568313 i,
 -0.0364413 - 0.0803933 i, -0.0157579 + 0.0110711 i, -0.0448814 - 0.0796178 i,
 -0.0309596 - 0.0620335 i, -0.0692182 - 0.151883 i, -0.0319339 - 0.00383458 i,
 -0.0233075 - 0.111581 i, -0.0101306 - 0.0822589 i, -0.0323024 - 0.21812 i,
 -0.0189583 - 0.0844727 i, -0.0556178 - 0.218061 i, -0.0195894 - 0.167156 i,
 0.373976 + 0.176453 i, 0.0481885 - 0.116956 i, 0.0218496 - 0.0050421 i,
 0.0379506 - 0.0147816 i, -0.0269679 - 0.025903 i, 0.0375501 - 0.00818858 i,
 -0.0296966 - 0.0290688 i, -0.0344551 - 0.0168195 i, -0.0071038 - 0.0670466 i,
 0.0193431 - 0.021617 i, -0.0122202 - 0.0328829 i, -0.0153121 - 0.016552 i,
 0.0102312 - 0.0824418 i, -0.01866 - 0.020663 i, 0.00323491 - 0.0849524 i,
 0.00804622 - 0.0662416 i, 0.0994447 + 0.109923 i, 0.0100959 - 0.000568641 i,
 -0.0268904 - 0.0495882 i, -0.0245207 - 0.0355702 i, -0.0215095 - 0.099105 i,
 -0.0280162 - 0.0379397 i, -0.0312304 - 0.0979598 i, -0.0138876 - 0.0789094 i,
 0.17717 - 0.0295302 i, -0.0106074 - 0.0458956 i, -0.00641741 - 0.133966 i,
 0.00722081 - 0.101594 i, 0.237598 + 0.0403602 i, -0.00222075 - 0.103981 i,
 0.2454 - 0.000476521 i, 0.171626 + 0.0649399 i, 0.462416 - 0.101295 i}
```

We must create a “working Qureg” **hψ** to compute expected values. We see our starting state isn’t especially close to the ground-state.

```
hψ = CreateQureg[nQ];
CalcExpecPauliSum[ψ, h, hψ]
```

```
-6.55243
```

We’ll also need additional Quregs to keep track of our fixed input state **inψ** = **|+⟩**, and **nθ** derivative states **dψ[i]** = $\left| \frac{\partial \psi}{\partial \theta[i]} \right\rangle$ (where **ψ = u(θ) inψ**)

```
inψ = CreateQureg[nQ];
InitPlusState[inψ];

dψ = CreateQuregs[nQ, nθ];
```

Computing variational observables

Let’s re-randomise the ansatz parameters.

```
vθ = Table[ θ[i] → RandomReal[],  {i, nθ}]
```

```
{θ[1] → 0.302214, θ[2] → 0.701127, θ[3] → 0.250832,
 θ[4] → 0.366587, θ[5] → 0.184505, θ[6] → 0.134426, θ[7] → 0.2041,
 θ[8] → 0.869532, θ[9] → 0.209075, θ[10] → 0.221281, θ[11] → 0.415592,
 θ[12] → 0.598776, θ[13] → 0.91903, θ[14] → 0.0748652, θ[15] → 0.824325,
 θ[16] → 0.0585614, θ[17] → 0.4362, θ[18] → 0.434674, θ[19] → 0.250115,
 θ[20] → 0.0151225, θ[21] → 0.71593, θ[22] → 0.408774, θ[23] → 0.793994,
 θ[24] → 0.943991, θ[25] → 0.711386, θ[26] → 0.455652, θ[27] → 0.0252188}
```

We set each **d$\boldsymbol{\psi}$[i] =** $\left|\frac{\partial u}{\partial\theta[i]}\ \mathtt{in}\psi\right\rangle$ for the given assignment of $\boldsymbol{\theta}$

```
CalcQuregDerivs[u, inψ, vθ,  dψ];
```

The imaginary time ‘tensor’ is simply **Re[** $\left\langle\frac{\partial\psi}{\partial\theta[i]}\middle|\middle|\frac{\partial\psi}{\partial\theta[j]}\right\rangle$ **]**

```
m = Re @ CalcInnerProducts[dψ];
m // Chop // MatrixForm
```

```
  0.25         0             -0.16127      0.25         0            -0.0458649
  0            0.25           0            0            0             0
 -0.16127      0              0.25        -0.16127      0             0.0295865    -
  0.25         0             -0.16127      0.25         0            -0.0458649
  0            0              0            0            0.25          0
 -0.0458649    0              0.0295865   -0.0458649    0             0.25         -0
  0.25         0             -0.16127      0.25         0            -0.0458649
  0            0              0            0            0             0
 -0.191007     0              0.123215    -0.191007     0             0.035042     -
  0.25         0             -0.16127      0.25         0            -0.0458649
  0            0              0            0            0             0
 -0.100933     0              0.0651097   -0.100933     0             0.0185171    -
  0.25         0             -0.16127      0.25         0            -0.0458649
  0            0              0            0            0             0
 -0.0186988    0              0.0120622   -0.0186988    0             0.00343048   -0
  0.25         0             -0.16127      0.25         0            -0.0458649
  0            0              0            0            0             0
 -0.105625     0              0.0681363   -0.105625     0             0.0193779    -
 -0.0512571    0.0213583     -0.0512571   -0.0512571   -0.00824045    0.0512571    -0
 -0.167895    -0.000900317    0.167895    -0.167895     0.00233352   -0.167895     -
 -0.0385019   -0.0124444      0.0385019   -0.0385019    0.00419934    0.00860778   -0
 -0.114527    -0.0181217      0.114527    -0.114527     0.00807583    0.0226566    -
 -0.14819     -0.0466023      0.14819     -0.14819      0.0293635     0.0392888    -
 -0.0793218   -0.0358798      0.0793218   -0.0793218    0.015727      0.0224444    -0
 -0.00813621   0.00265984     0           -0.00813621   0.00320796    0            -0
  0.039606    -0.0140054      0            0.039606    -0.00428338    0             0
  0.0002884    0             -0.000447077  0.0002884    0            -0.00157201    0
```

We set $|\boldsymbol{\psi}\rangle = \mathbf{u}(\boldsymbol{\theta})\ |\mathbf{in}\boldsymbol{\psi}\rangle$, and $|\mathbf{h}\boldsymbol{\psi}\rangle = \mathbf{h}\ |\boldsymbol{\psi}\rangle$

```
ApplyCircuit[u /. vθ,  CloneQureg[ψ, inψ]];
ApplyPauliSum[ψ, h, hψ];
```

The energy gradient is then simply **Re[** $\left\langle\boldsymbol{\psi}\middle|\mathbf{h}\middle|\frac{\partial\psi}{\partial\theta[i]}\right\rangle$ **]**

```
v = Re @ CalcInnerProducts[hψ, dψ];
v // MatrixForm
```

$$\begin{pmatrix} -0.013409 \\ -0.00539944 \\ 0.0379003 \\ -0.013409 \\ -0.0105197 \\ 0.0604042 \\ -0.013409 \\ 0.00468087 \\ 0.0250343 \\ -0.013409 \\ -0.0564997 \\ 0.00145499 \\ -0.013409 \\ -0.0671572 \\ -0.0213131 \\ -0.013409 \\ -0.0855882 \\ 0.0411782 \\ -0.0196398 \\ -0.0389648 \\ -0.010845 \\ 0.022991 \\ 0.0449788 \\ 0.000658498 \\ 0.00793236 \\ -0.0329018 \\ 0.018517 \end{pmatrix}$$

A single iteration of imaginary time evolution would use these observables to update the parameters via $\mathbf{m}\,\Delta\boldsymbol{\theta} = -\mathbf{v}\,\Delta\mathbf{t}$

```
Δθ = Δt LinearSolve[m, -v];
Δθ // MatrixForm
```

$$\begin{pmatrix} -0.112232\,\Delta t \\ -0.247915\,\Delta t \\ 1.37134\,\Delta t \\ -0.112232\,\Delta t \\ 0.191551\,\Delta t \\ -1.42699\,\Delta t \\ -0.112232\,\Delta t \\ -0.0810106\,\Delta t \\ 0.0389451\,\Delta t \\ -0.112232\,\Delta t \\ 0.408794\,\Delta t \\ -0.26508\,\Delta t \\ -0.112232\,\Delta t \\ 0.501992\,\Delta t \\ -0.0693181\,\Delta t \\ -0.112232\,\Delta t \\ 0.741091\,\Delta t \\ -0.583384\,\Delta t \\ 1.50634\,\Delta t \\ -1.15593\,\Delta t \\ 0.414318\,\Delta t \\ -0.365748\,\Delta t \\ -0.275681\,\Delta t \\ -0.706214\,\Delta t \\ -0.0677737\,\Delta t \\ 0.379378\,\Delta t \\ -0.0555886\,\Delta t \end{pmatrix}$$

hence the updated parameters, dependent on **Δ**t**,** would be

```
vθ⟦All, 2⟧ += Δθ
```

```
{0.302214 - 0.112232 Δt, 0.701127 - 0.247915 Δt, 0.250832 + 1.37134 Δt,
 0.366587 - 0.112232 Δt, 0.184505 + 0.191551 Δt, 0.134426 - 1.42699 Δt,
 0.2041 - 0.112232 Δt, 0.869532 - 0.0810106 Δt, 0.209075 + 0.0389451 Δt,
 0.221281 - 0.112232 Δt, 0.415592 + 0.408794 Δt, 0.598776 - 0.26508 Δt,
 0.91903 - 0.112232 Δt, 0.0748652 + 0.501992 Δt, 0.824325 - 0.0693181 Δt,
 0.0585614 - 0.112232 Δt, 0.4362 + 0.741091 Δt, 0.434674 - 0.583384 Δt,
 0.250115 + 1.50634 Δt, 0.0151225 - 1.15593 Δt, 0.71593 + 0.414318 Δt,
 0.408774 - 0.365748 Δt, 0.793994 - 0.275681 Δt, 0.943991 - 0.706214 Δt,
 0.711386 - 0.0677737 Δt, 0.455652 + 0.379378 Δt, 0.0252188 - 0.0555886 Δt}
```

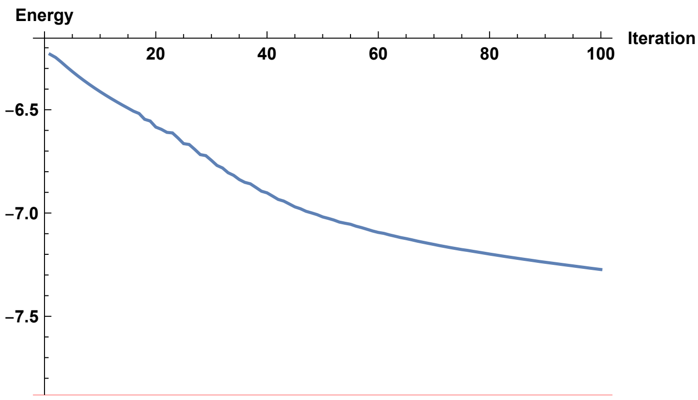
Performing variational imaginary time

We simply repeat these steps to perform variational imaginary time minimization. We'll measure and record the energy at each iteration.

```
vθ = Table[ θ[i] → RandomReal[],  {i, nθ}];

Δt = .01;
nt = 100;

en = Table[
       (* compute imaginary time tensor *)
       CalcQuregDerivs[u, inψ, vθ, dψ];
       m = Re @ CalcInnerProducts[dψ];

       (* compute energy gradient *)
       ApplyCircuit[u /. vθ, CloneQureg[ψ, inψ]];
       ApplyPauliSum[ψ, h, hψ];
       v = Re @ CalcInnerProducts[hψ, dψ];

       (* update parameters under imaginary time *)
       Δθ = Δt LinearSolve[m, -v];
       vθ[[All, 2]] += Δθ;

       (* record the current energy *)
       CalcExpecPauliSum[ψ, h, hψ],

       (* perform nt iterations *)
       nt
  ];
```

Here's how the energy evolved

```
ListLinePlot[en,
    GridLines → {{}, {{gs, Red}}},
    PlotRange → {gs, Automatic},
    AxesLabel → {"Iteration", "Energy"}]
```



and the final energy, and its corresponding parameters

```
Row[{Last[en], "  /  ", gs}]
vθ
```

```
-7.27344  /  -7.88074
```

```
{θ[1] → 0.310389, θ[2] → 0.0898798, θ[3] → 1.48073, θ[4] → 0.0527977,
 θ[5] → 0.801902, θ[6] → 0.262244, θ[7] → 0.722303, θ[8] → 1.53805,
 θ[9] → 0.913898, θ[10] → 0.26469, θ[11] → 1.53865, θ[12] → -0.483215,
 θ[13] → 0.0282503, θ[14] → 1.54369, θ[15] → 0.687921, θ[16] → 0.584671,
 θ[17] → 1.52099, θ[18] → 0.533196, θ[19] → 0.826099, θ[20] → -0.714602,
 θ[21] → -0.401971, θ[22] → 0.00317968, θ[23] → -0.189258,
 θ[24] → -0.157878, θ[25] → 1.5827, θ[26] → 1.0375, θ[27] → 0.231741}
```