definitions of quantum gates

single-qubit pseudo spin operators: $\mathbb{1}, \hat{\sigma}_x, \hat{\sigma}_y, \hat{\sigma}_z$

```
In[∘]:= {id, σx, σy, σz} = Table[SparseArray[PauliMatrix[i]], {i, 0, 3}];
```

projectors onto 0 and 1 states, respectively:

```
In[∘]:= proj0 = (id + σz)/2 // SparseArray;
        proj1 = (id - σz)/2 // SparseArray;
```

operator $\hat{a}$ acting on the k^{th} qubit of a set of n qubits: $\mathbb{1}\otimes\mathbb{1}\otimes\cdots\otimes\mathbb{1}\otimes\hat{a}\otimes\mathbb{1}\otimes\cdots\otimes\mathbb{1}\otimes\mathbb{1}$

```
In[∘]:= op[n_Integer, k_Integer, a_] /; 1 ≤ k ≤ n ∧ Dimensions[a] == {2, 2} :=
         KroneckerProduct[IdentityMatrix[2^(k-1), SparseArray],
          a, IdentityMatrix[2^(n-k), SparseArray]]
```

controlled operator: in a set of n qubits, only if all of the qubits in the list λ are in state 1 then the operator $\hat{A}$ is applied to the system; otherwise no operation is applied

```
In[∘]:= ctrl[n_Integer, λ_ /; VectorQ[λ, IntegerQ], A_] /;
           (Unequal @@ λ) ∧ Min[λ] ≥ 1 ∧ Max[λ] ≤ n ∧ Dimensions[A] == {2^n, 2^n} :=
         IdentityMatrix[2^n, SparseArray] +
          Apply[Dot, op[n, #, proj1] & /@ λ].(A - IdentityMatrix[2^n, SparseArray])
```

single-qubit gates

Pauli gates acting on the k^{th} qubit of a set of n qubits:

```
In[∘]:= xGate[n_Integer, k_Integer] /; 1 ≤ k ≤ n := op[n, k, σx]
        yGate[n_Integer, k_Integer] /; 1 ≤ k ≤ n := op[n, k, σy]
        zGate[n_Integer, k_Integer] /; 1 ≤ k ≤ n := op[n, k, σz]
```

Pauli rotations: $\hat{R}_x(\phi) \propto e^{-i\,\phi\,\hat{\sigma}_x/2}$ etc.

```
In[∘]:= rxGate[n_Integer, k_Integer, ϕ_] /; 1 ≤ k ≤ n := op[n, k, (1 + E^(I ϕ))/2 id + (1 - E^(I ϕ))/2 σx]
        ryGate[n_Integer, k_Integer, ϕ_] /; 1 ≤ k ≤ n := op[n, k, (1 + E^(I ϕ))/2 id + (1 - E^(I ϕ))/2 σy]
        rzGate[n_Integer, k_Integer, ϕ_] /; 1 ≤ k ≤ n := op[n, k, (1 + E^(I ϕ))/2 id + (1 - E^(I ϕ))/2 σz]
```

Hadamard gate:

```
In[∘]:= hadamardGate[n_Integer, k_Integer] /; 1 ≤ k ≤ n := op[n, k, (σx + σz)/Sqrt[2]]
```

two-qubit gates

SWAP gate: exchanges the state of qubits j and k in a set of n qubits

In[•]:=
```
swapGate[n_Integer, {j_Integer, k_Integer}] /; 1 ≤ j ≤ n ∧ 1 ≤ k ≤ n ∧ j ≠ k :=
  (IdentityMatrix[2^n, SparseArray] + xGate[n, j].xGate[n, k] +
     yGate[n, j].yGate[n, k] + zGate[n, j].zGate[n, k]) / 2
```

square root of the SWAP gate:

In[•]:=
```
sqrtSwapGate[n_Integer, {j_Integer, k_Integer}] /; 1 ≤ j ≤ n ∧ 1 ≤ k ≤ n ∧ j ≠ k :=
  (3 + ⅈ)/4 IdentityMatrix[2^n, SparseArray] +
   (1 - ⅈ)/4 (xGate[n, j].xGate[n, k] + yGate[n, j].yGate[n, k] + zGate[n, j].zGate[n, k])
```

CNOT gate: in a set of n qubits, if qubit j is in state 0 then there is no effect on qubit k, whereas if qubit j is in state 1 then the NOT operator $\hat{\sigma}_x$ acts on qubit k

In[•]:=
```
cnotGate[n_Integer, j_Integer → k_Integer] /; 1 ≤ j ≤ n ∧ 1 ≤ k ≤ n ∧ j ≠ k :=
  ctrl[n, {j}, op[n, k, σx]]
```

three-qubit gates

CCNOT (Toffoli) gate: in a set of n qubits, if both qubits i and j are in state 1, then the NOT operator $\hat{\sigma}_x$ acts on qubit k

In[•]:=
```
ccnotGate[n_Integer, {i_Integer, j_Integer} → k_Integer] /;
   1 ≤ i ≤ n ∧ 1 ≤ j ≤ n ∧ 1 ≤ k ≤ n ∧ Unequal[i, j, k] := ctrl[n, {i, j}, op[n, k, σx]]
```

controlled SWAP (Fredkin) gate: in a set of n qubits, if qubit i is in state 0 then there is no effect on qubits j and k, whereas if qubit i is in state 1 then the state of qubits j and k is exchanged

In[•]:=
```
cswapGate[n_Integer, i_Integer → {j_Integer, k_Integer}] /;
   1 ≤ i ≤ n ∧ 1 ≤ j ≤ n ∧ 1 ≤ k ≤ n ∧ Unequal[i, j, k] := ctrl[n, {i}, swapGate[n, {j, k}]]
```

a simple circuit

the operator for the circuit:

In[•]:=
```
s = cnotGate[2, 1 → 2].hadamardGate[2, 1];
Normal[s]
```

Out[•]= $\left\{\left\{\frac{1}{\sqrt{2}}, 0, \frac{1}{\sqrt{2}}, 0\right\}, \left\{0, \frac{1}{\sqrt{2}}, 0, \frac{1}{\sqrt{2}}\right\}, \left\{0, \frac{1}{\sqrt{2}}, 0, -\frac{1}{\sqrt{2}}\right\}, \left\{\frac{1}{\sqrt{2}}, 0, -\frac{1}{\sqrt{2}}, 0\right\}\right\}$

the basis set:

In[•]:=
```
bas[n_Integer /; n ≥ 1] := Tuples[{0, 1}, n]
bas[2]
```

Out[•]= {{0, 0}, {0, 1}, {1, 0}, {1, 1}}

operate on the $|00\rangle$ input state:

In[•]:=
```
ψin = {1, 0, 0, 0};
ψout = s.ψin
```

Out[•]= $\left\{\frac{1}{\sqrt{2}}, 0, 0, \frac{1}{\sqrt{2}}\right\}$

Quantum Fourier Transform

The Quantum Fourier Transform (QFT) is assembled following Fig. 5.1 of M. Nielsen and I. Chuang, *Quantum Computation and Quantum Information*, Cambridge University Press, 2010 (tenth anniversary edition).

The quantum circuit for a QFT is assembled from blocks that connect the i^{th} qubit to the qubits $\{i+1, i+2, \ldots, n\}$ via controlled phase gates:

In[•]:=
```
qftBlock[n_Integer, i_Integer] /; 1 ≤ i ≤ n :=
 Apply[Dot, Table[ctrl[n, {j}, rzGate[n, i, 2 π / 2^(j+1-i)]], {j, n, i + 1, -1}]].
  hadamardGate[n, i]
```

assemble the quantum Fourier transformation by connecting all qubits:

1. connect all qubits through the above QFTblock operation
2. swap the order of the qubits

In[•]:=
```
qft[n_Integer] /; n ≥ 1 := Apply[Dot, Table[swapGate[n, {i, n + 1 - i}], {i, 1, n/2}]].
  Apply[Dot, Table[qftBlock[n, i], {i, n, 1, -1}]]
```

QFT consists of a polynomial number of gates:

· n Hadamard gates
· $\frac{n(n-1)}{2}$ controlled rotations
· $\left\lfloor \frac{n}{2} \right\rfloor$ swap gates

simple formula for the resulting matrix representation:

In[•]:=
```
Table[qft[n] == 1/2^(n/2) Table[ⅇ^(2 π ⅈ j k/2^n), {j, 0, 2^n - 1}, {k, 0, 2^n - 1}], {n, 6}] //
 FullSimplify
```

Out[•]=
```
{True, True, True, True, True, True}
```

Quantum phase estimation

estimate the phase of this:

In[•]:=
```
w = {1};
u[ϕ_] = {{ⅇ^(2 π ⅈ ϕ)}};
```

check that we set up the problem correctly:

In[•]:=
```
{u[ϕ].w === ⅇ^(2 π ⅈ ϕ) w, Norm[w] == 1}
```

Out[•]=
```
{True, True}
```

unit operator in the space of the system U:

In[•]:=
```
u0 = IdentityMatrix[Length[w], SparseArray];
```

U operator attached to n qubits, controlled by the i^{th} qubit:

In[•]:=
```
ctrlU[n_Integer, i_Integer, ϕ_] /; 1 ≤ i ≤ n :=
 KroneckerProduct[op[n, i, proj0], u0] + KroneckerProduct[op[n, i, proj1], u[ϕ]]
```

estimate to t digits of precision:

In[•]:=
```
t = 4;
ϵ[ϕ_] = KroneckerProduct[ConjugateTranspose[qft[t]], u0].
    Apply[Dot, Table[ctrlU[t, i, 2^(t-i) ϕ], {i, t, 1, -1}]].
    KroneckerProduct[Apply[Dot, Table[hadamardGate[t, i], {i, t}]], u0].
    Flatten[KroneckerProduct[SparseArray[1 → 1, 2^t], w]] // Normal;
```

probabilities for measuring the different basis states: trace out the SUT and look at the diagonal elements of the reduced density matrix
(see ReducedDensityMatrix.nb)

In[•]:=
```
rdm[ψABC_?VectorQ, {dA_Integer /; dA ≥ 1, dB_Integer /; dB ≥ 1,
    dC_Integer /; dC ≥ 1}] /; Length[ψABC] == dA dB dC :=
 With[{P = ArrayReshape[ψABC, {dA, dB, dC}]},
  Flatten[Transpose[P, {1, 3, 2}].ConjugateTranspose[P], {{1, 2}, {3, 4}}]]
traceout[ψ_?VectorQ, d_Integer /; d ≥ 1] /; Divisible[Length[ψ], d] :=
 rdm[ψ, {1, d, Length[ψ]/d}]
traceout[ψ_?VectorQ, d_Integer /; d ≤ -1] /; Divisible[Length[ψ], -d] :=
 rdm[ψ, {Length[ψ]/(-d), -d, 1}]
```

In[•]:=
```
prob[ϕ_?NumericQ] := Re[Diagonal[traceout[ϵ[N[ϕ]], -Length[w]]]]
```

When ϕ is an integer multiple of 2^{-t}, only one basis state has 100% probability of occurring. The i^{th} basis state corresponds to a measurement of $\phi = \frac{i-1}{2^t}$:

In[•]:=
```
Table[prob[ϕ], {ϕ, 0, 1, 2^-t}] // Chop // MatrixForm
```

Out[•]//MatrixForm=

$$\begin{pmatrix}
1. & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1. & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1. & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1. & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1. & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1. & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1. & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1. & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1. & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1. & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1. & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1. & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1. & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1. & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1. & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1. \\
1. & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}$$

When ϕ is not an integer multiple of 2^{-t}, all basis states can occur in measurement:

In[•]:=
```
Round[prob[0.2], 0.001]
```

Out[•]=
```
{0.004, 0.008, 0.025, 0.876, 0.055, 0.011, 0.005,
 0.003, 0.002, 0.002, 0.001, 0.001, 0.001, 0.002, 0.002, 0.003}
```

The mean measurement is a bad estimator (doesn't converge as $t \to \infty$).
The most likely measurement is a good estimator.

In[◦]:=
```
mean[ϕ_?NumericQ] := prob[ϕ] . Range[0, 2^t - 1]/2^t;
mostprobable[ϕ_?NumericQ] := (Ordering[prob[ϕ], -1][[1]] - 1)/2^t;
Plot[{ϕ, mean[ϕ], mostprobable[ϕ]}, {ϕ, 0, 1},
 AxesLabel → {"phase setting", "phase measurement"},
 PlotRange → {{0, 1}, {0, 1}}, PlotRangePadding → None,
 PlotStyle → {Black, {Thick, Blue}, {Thick, Red}}, AspectRatio → {1},
 PlotLegends → {"exact", "mean of measurements", "most frequent measurement"},
 GridLines → {Range[1, 2^t - 1]/2^t, Range[1, 2^t - 1]/2^t}]
```

Out[◦]=