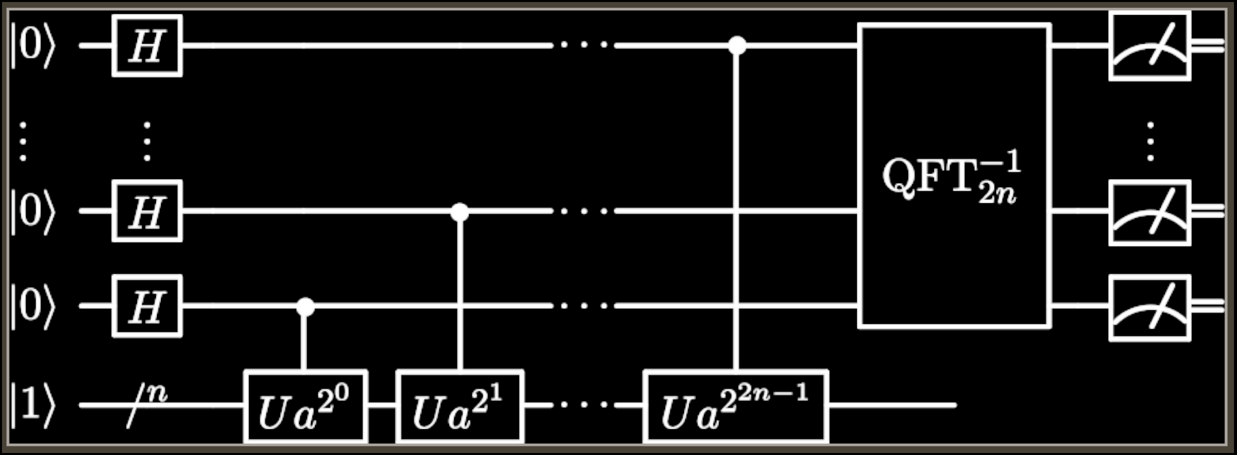


Dynamika układów kwantowych i wprowadzenie do pakietu QuTiP (Quantum Toolbox in Python)

Paweł Szumniak

Wydział Fizyki i Informatyki Stosowanej Akademia Górniczo-Hutnicza w Krakowie

Fundusze Europejskie Inteligentny Rozwój
Rzeczpospolita Polska
FNP Fundacja na rzecz Nauki Polskiej
Unia Europejska Europejski Fundusz Rozwoju Regionalnego

Ewolucja czasowa układów kwantowych

Zależne od czasu równanie Schroedingera

$$i\hbar\frac{\partial\Psi(\mathbf{r},t)}{\partial t} = H\Psi(\mathbf{r},t) = H\begin{pmatrix}\psi_\uparrow(\mathbf{r},t)\\ \psi_\downarrow(\mathbf{r},t)\end{pmatrix}$$

Separacja części przestrzennej i spinowej/orbitalnej

$$\Psi(\mathbf{r},t) = \begin{pmatrix}\psi_\uparrow(\mathbf{r},t)\\ \psi_\downarrow(\mathbf{r},t)\end{pmatrix} = \Psi(\mathbf{r})\begin{pmatrix}\alpha_\uparrow(t)\\ \alpha_\downarrow(t)\end{pmatrix}$$

Ewolucja czasowa układów kwantowych

Zależne od czasu równanie Schroedingera

$$i\hbar\frac{d}{dt}|\Psi(t)\rangle = \hat{H}(t)|\Psi(t)\rangle$$

Dla pojedynczego kubitu

$$|\Psi(t)\rangle = a(t)|0\rangle + b(t)|1\rangle = \begin{pmatrix} a(t) \\ b(t) \end{pmatrix} \quad \hat{H} = \begin{pmatrix} h_{11}(t) & h_{12}(t) \\ h_{21}(t) & h_{22}(t) \end{pmatrix}$$

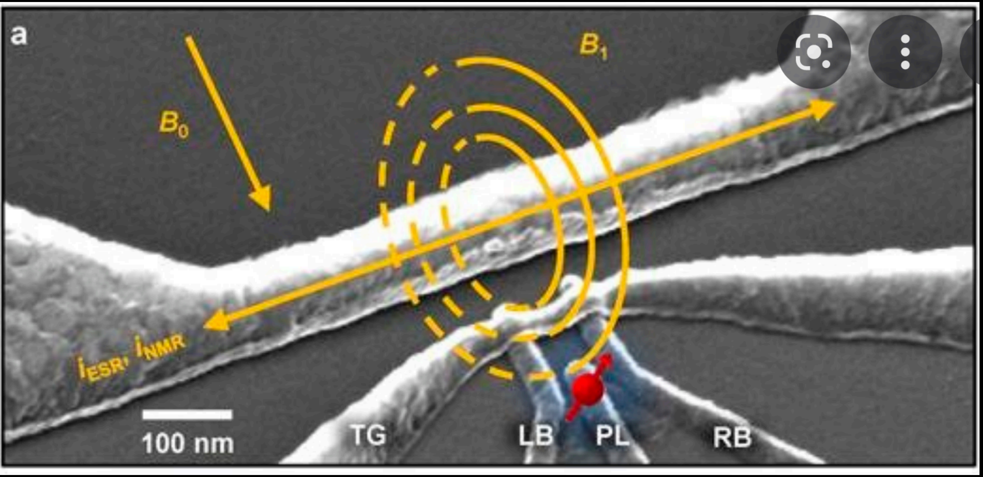
Układ równań róćzniczkowych zwyczajnych na a(t) i b(t)

$$\begin{cases} i\hbar\dot{a}(t) = h_{11}(t)a(t) + h_{12}(t)b(t) \\ i\hbar\dot{b}(t) = h_{21}(t)a(t) + h_{22}(t)b(t) \end{cases}$$

Przykład Oscylacje Rabiego – Rezonans NMR (Nuclear Magnetic Resonance)

$$\hat{H} = -\vec{\mu}\cdot\vec{B}$$

$$\vec{\mu} = \frac{1}{2}\gamma_p\vec{\sigma}, \quad \gamma_p = 5.59\frac{q_p\hbar}{2m_p}$$



$$\vec{B}_1(t) = (\cos(\omega t), -\sin(\omega t), 0)$$

$$\vec{B}_0(t) = (0, 0, B_0)$$

$$\vec{B}(t) = (\cos(\omega t), -\sin(\omega t), B_0)$$

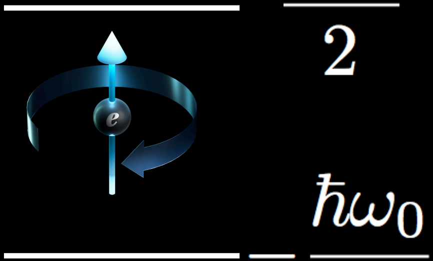
$$\hat{H}(t) = -\frac{1}{2}\gamma_p B_0\sigma_z - \frac{1}{2}\gamma_p B_1\left(\cos(\omega t)\sigma_x - \sin(\omega t)\sigma_y\right)$$

Przykład Oscylacje Rabiego – Rezonans NMR (Nuclear Magnetic Resonance)

$$\hat{H}(t) = -\frac{1}{2}\gamma_p B_0 \sigma_z - \frac{1}{2}\gamma_p B_1 \left(\cos(\omega t)\sigma_x - \sin(\omega t)\sigma_y\right)$$

$\hbar\omega_0 = \gamma_p B_0$ $\hbar\omega_1 = \gamma_p B_1$

$$\frac{\hbar\omega_0}{2} \quad |1\rangle = |\downarrow\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$



$$-\frac{\hbar\omega_0}{2} \quad |0\rangle = |\uparrow\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$\hat{H}(t) = -\frac{\hbar}{2}\omega_0 \sigma_z - \frac{\hbar}{2}\omega_1 \left(\cos(\omega t)\sigma_x - \sin(\omega t)\sigma_y\right)$$

Przykład Oscylacje Rabiego – Rezonans NMR (Nuclear Magnetic Resonance)

$$\hat{H}(t) = -\frac{\hbar}{2}\begin{pmatrix} \omega_0 & \omega_1 e^{i\omega t} \\ \omega_1 e^{-i\omega t} & -\omega_0 \end{pmatrix}$$

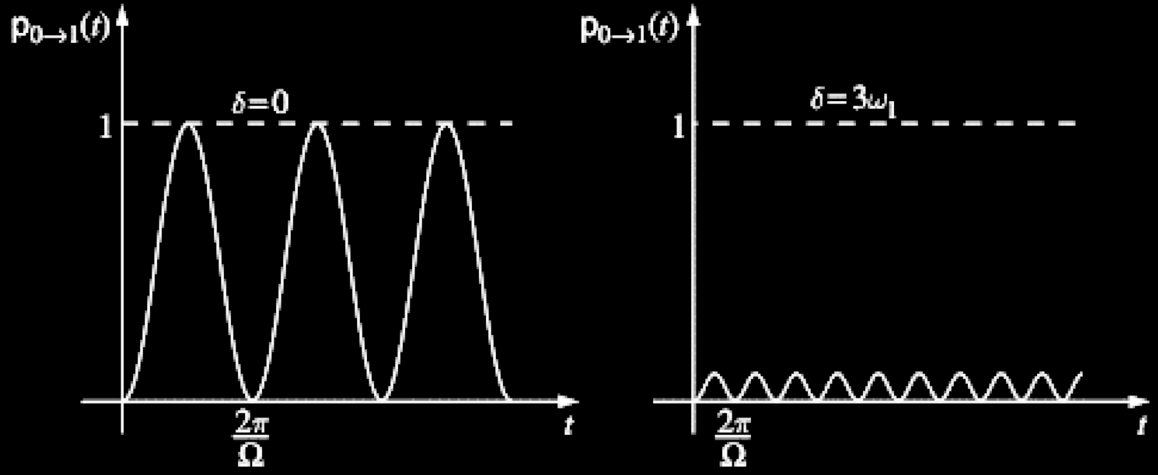
$$|\Psi(t)\rangle = a(t)|0\rangle + b(t)|1\rangle = \begin{pmatrix} a(t) \\ b(t) \end{pmatrix}$$

$$P_{|0\rangle\to|1\rangle}(t) = \left(\frac{\omega_1}{\Omega}\right)^2 \sin^2\left(\frac{\Omega t}{2}\right) \qquad \Omega = \sqrt{(\omega-\omega_0)^2+\omega_1^2}$$

Przykład Oscylacje Rabiego – Rezonans NMR (Nuclear Magnetic Resonance)

$$P_{|0\rangle\to|1\rangle}(t) = \left(\frac{\omega_1}{\Omega}\right)^2 \sin^2\left(\frac{\Omega t}{2}\right) \qquad \Omega = \sqrt{(\omega-\omega_0)^2+\omega_1^2}$$

$$P_{|0\rangle\to|1\rangle}(t) = \sin^2\left(\frac{\omega_1 t}{2}\right), \quad \omega = \omega_0$$



$$\frac{\omega_1 t}{2} = \frac{\pi}{2}, \quad t = \frac{\pi}{\omega_1}$$

$$\frac{\omega_1 t}{2} = \frac{\pi}{4}, \quad t = \frac{\pi}{2\omega_1}$$

Figure 3.4 Rabi oscillations. The *detuning* δ is defined as $\delta = \omega - \omega_0$.

Ewolucja czasowa układów kwantowych

Zależne od czasu równanie Schroedingera

$$i\hbar\frac{d}{dt}|\Psi(t)\rangle = \hat{H}(t)|\Psi(t)\rangle$$

Dla n kubitów

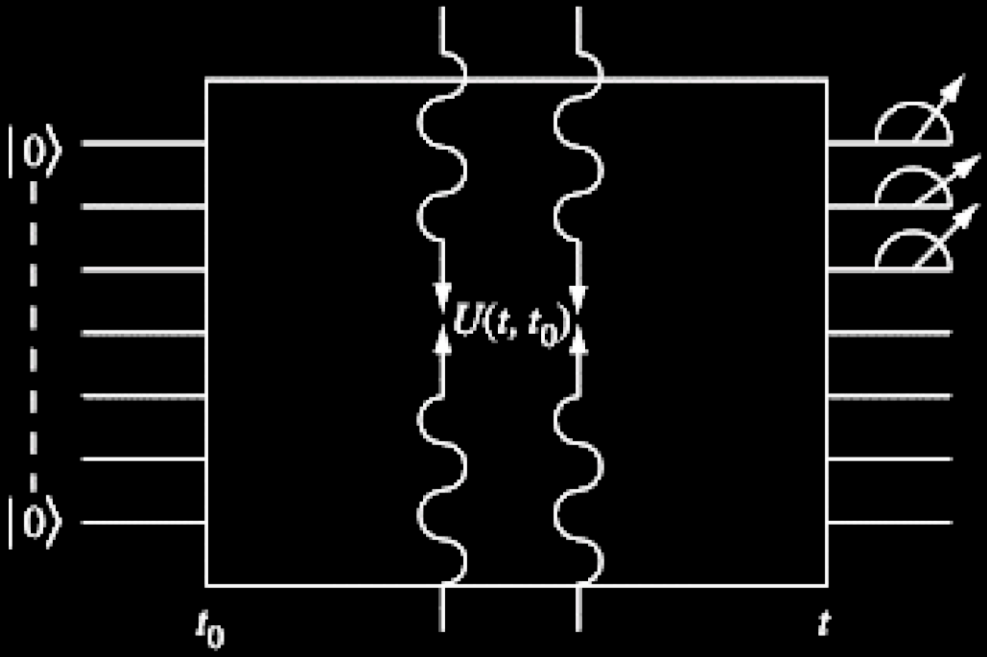
$$|\Psi(t)\rangle = \sum_{j=1}^{2^n} a_j|j\rangle$$

Układ 2^n równań różniczkowych zwyczajnych na $a_i(t)$, i=1,2, …, 2^n

$$i\hbar\begin{pmatrix}\dot{a}_1(t)\\ \dot{a}_2(t)\\ \vdots\\ \dot{a}_{2^n}(t)\end{pmatrix} = \begin{pmatrix}h_{11}(t) & h_{12}(t) & \dots & h_{12^n}(t)\\ h_{21}(t) & h_{22}(t) & \dots & h_{22^n}(t)\\ \vdots & \vdots & \ddots & \vdots\\ h_{2^n1}(t) & h_{2^n2}(t) & \dots & h_{2^n2^n}(t)\end{pmatrix}\begin{pmatrix}a_1(t)\\ a_2(t)\\ \vdots\\ a_{2^n}(t)\end{pmatrix}$$

Operator ewolucji czasowej, a bramki kwantowe

$$\hat{U}(t,t_0) = \exp\left[-\frac{i}{\hbar}\int_{t_0}^{t} dt'\,\hat{H}(t')\right]$$



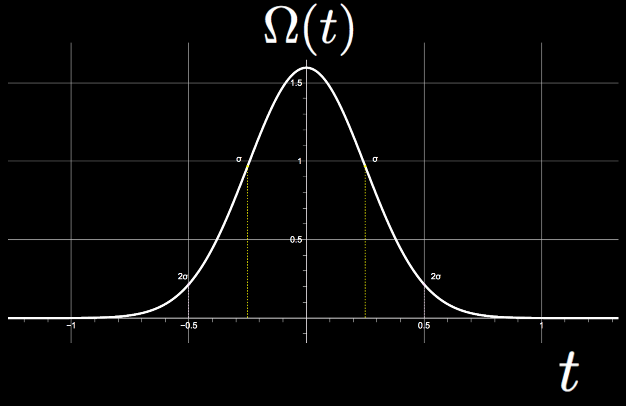
Operator ewolucji czasowej - impulsy generujące kwantowe bramki logiczne obrotu

$$\hat{U}(t,t_0) = \exp\left[-\frac{i}{\hbar}\int_{t_0}^{t} dt'\,\hat{H}(t')\right]$$

$$-\boxed{R_{\vec{n}}(\theta)}-$$

Hamiltonian cząstki o spinie ½ w polu magnetycznym **B**:

$$\hat{H}(t) = g\mu_B\vec{B}(t)\cdot\vec{S} = \frac{\hbar}{2}g\mu_B B_0(t)\vec{n}\cdot\vec{\sigma} = \frac{\hbar\Omega(t)}{2}\vec{n}\cdot\vec{\sigma}$$



$$\hat{U}(t,t_0) = \exp\left[-i\underbrace{\frac{1}{\hbar}\int_{-\infty}^{\infty} dt'\,\frac{\hbar\Omega(t')}{2}}\vec{n}\cdot\vec{\sigma}\right]$$

$$R_{\vec{n}}(\theta) \equiv \exp\left(-i\vec{n}\cdot\vec{\sigma}\boxed{\frac{\theta}{2}}\right) = I\cos(\theta/2) - i(\vec{n}\cdot\vec{\sigma})\sin(\theta/2)$$

Macierz gestości

$$\rho(t) = \begin{pmatrix} \rho_{00}(t) & \rho_{01}(t) \\ \rho_{10}(t) & \rho_{11}(t) \end{pmatrix}$$

Stany czyste

$$\rho(t) = |\Psi(t)\rangle\langle\Psi(t)| = \begin{pmatrix} |\alpha_0(t)|^2 & \alpha_0^*(t)\alpha_1(t) \\ \alpha_0(t)\alpha_1^*(t) & |\alpha_1(t)|^2 \end{pmatrix} \quad \rho^2 = \rho$$

Stany mieszane

$$\rho = \sum_j \lambda_j |j\rangle\langle j|$$

Liouville–von Neumann equations

$$\frac{d}{dt}\rho(t) = -\frac{i}{\hbar}[H, \rho(t)]$$

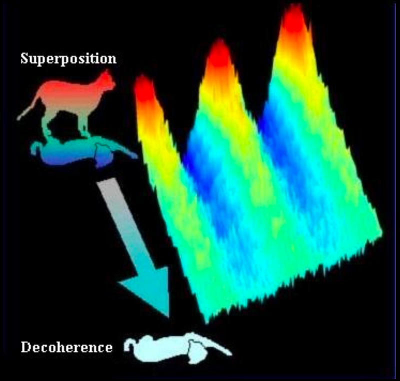
The Lindblad Master equation - dekoherencia

$$\dot{\rho}_{\text{tot}}(t) = -\frac{i}{\hbar}[H_{\text{tot}}, \rho_{\text{tot}}(t)]$$

$$H_{\text{tot}} = H_{\text{sys}} + H_{\text{env}} + H_{\text{int}}$$

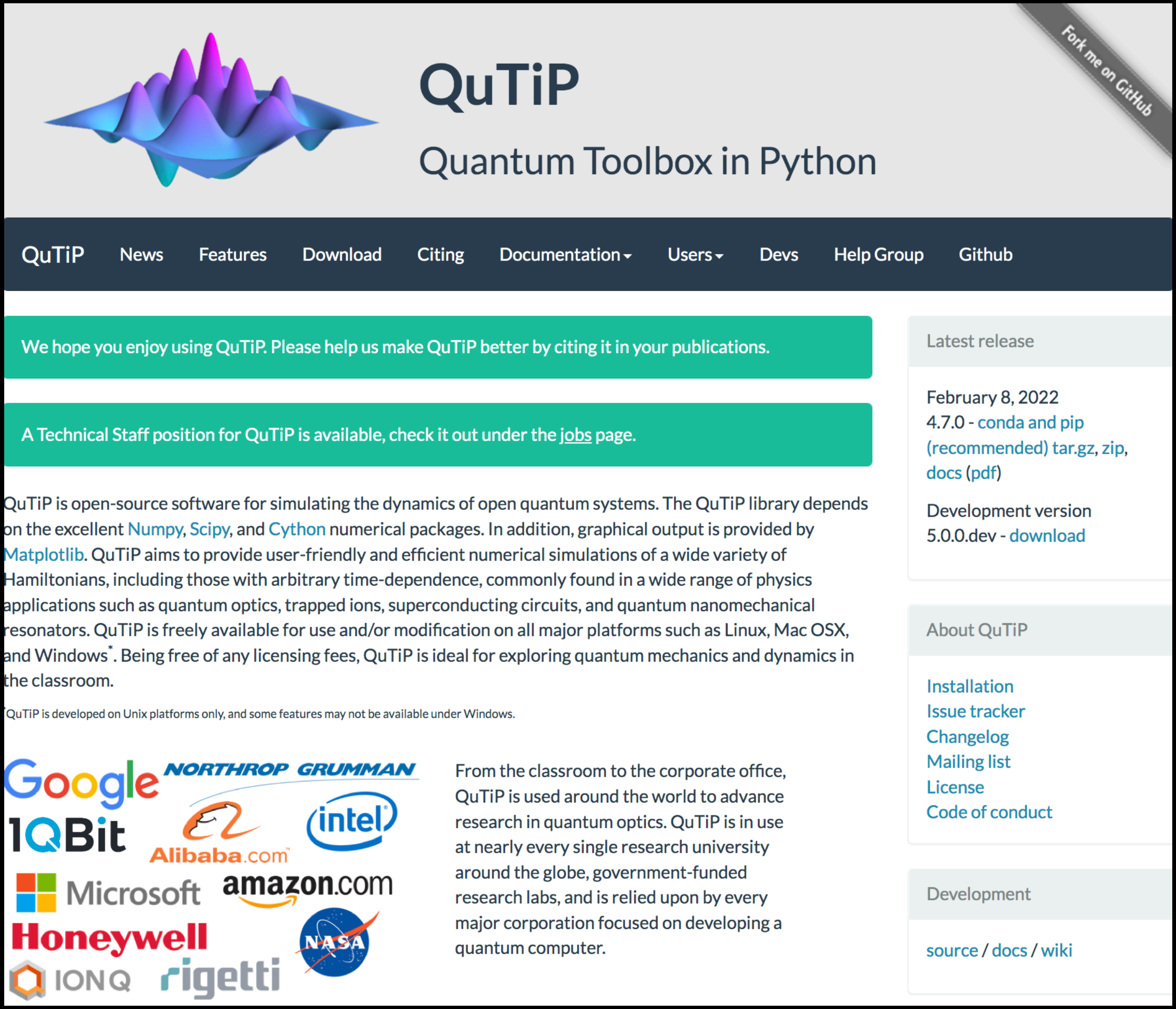
$$\rho = \text{Tr}_{\text{env}}[\rho_{\text{tot}}] \qquad \rho_{\text{tot}}(t) \approx \rho(t) \otimes \rho_{\text{env}}$$

$$\dot{\rho}(t) = -\frac{i}{\hbar}[H(t), \rho(t)] + \sum_n \frac{1}{2}\left[2C_n\rho(t)C_n^\dagger - \rho(t)C_n^\dagger C_n - C_n^\dagger C_n\rho(t)\right]$$



$$C_n = \sqrt{\gamma_n}A_n$$

QuTiP https://qutip.org/



QuTiP

Quantum Toolbox in Python

QuTiP News Features Download Citing Documentation▾ Users▾ Devs Help Group Github

We hope you enjoy using QuTiP. Please help us make QuTiP better by citing it in your publications.

A Technical Staff position for QuTiP is available, check it out under the jobs page.

QuTiP is open-source software for simulating the dynamics of open quantum systems. The QuTiP library depends on the excellent Numpy, Scipy, and Cython numerical packages. In addition, graphical output is provided by Matplotlib. QuTiP aims to provide user-friendly and efficient numerical simulations of a wide variety of Hamiltonians, including those with arbitrary time-dependence, commonly found in a wide range of physics applications such as quantum optics, trapped ions, superconducting circuits, and quantum nanomechanical resonators. QuTiP is freely available for use and/or modification on all major platforms such as Linux, Mac OSX, and Windows*. Being free of any licensing fees, QuTiP is ideal for exploring quantum mechanics and dynamics in the classroom.

*QuTiP is developed on Unix platforms only, and some features may not be available under Windows.

From the classroom to the corporate office, QuTiP is used around the world to advance research in quantum optics. QuTiP is in use at nearly every single research university around the globe, government-funded research labs, and is relied upon by every major corporation focused on developing a quantum computer.

Latest release

February 8, 2022
4.7.0 - conda and pip (recommended) tar.gz, zip, docs (pdf)

Development version
5.0.0.dev - download

About QuTiP

Installation
Issue tracker
Changelog
Mailing list
License
Code of conduct

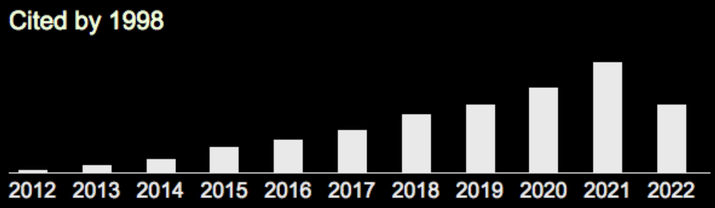
Development

source / docs / wiki

QuTiP Contributors and papers

J. R. Johansson, P. D. Nation, and F. Nori: "QuTiP 2: A Python framework for the dynamics of open quantum systems.", Comp. Phys. Comm. 184, 1234 (2013) [DOI: 10.1016/j.cpc.2012.11.019].

J. R. Johansson, P. D. Nation, and F. Nori: "QuTiP: An open-source Python framework for the dynamics of open quantum systems.", Comp. Phys. Comm. 183, 1760–1772 (2012) [DOI: 10.1016/j.cpc.2012.02.021].



Paul Nation
IBM Q
Library designer and main contributor

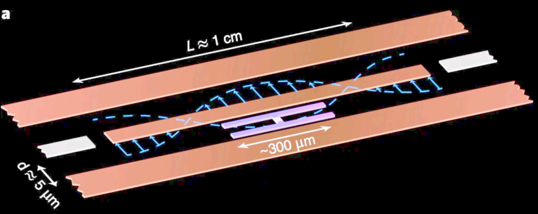
Robert Johansson
Tokyo, Japan
Library designer and main contributor

Franco Nori
RIKEN / University of Michigan

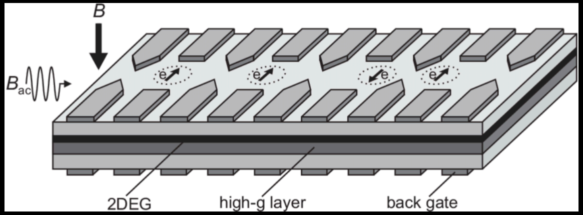
https://qutip.org/devs.html

Układy fizyczne, których dynamikę można symulować przy uzyciu pakietu QuTiP

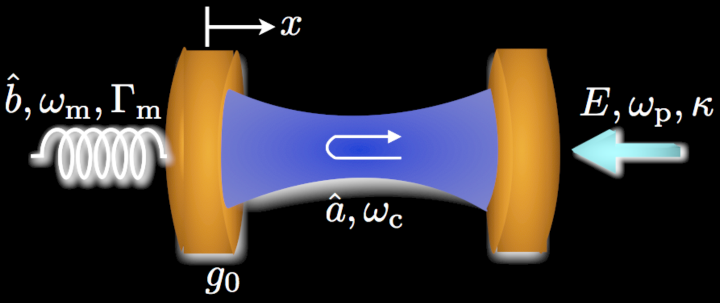
circuit QED



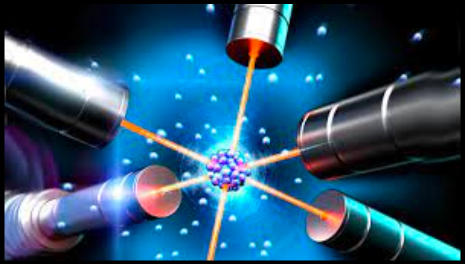
Spin chain



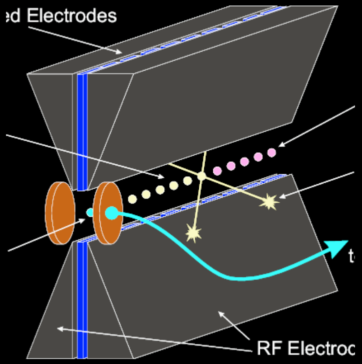
Optomechanical Systems


x
$\hat{b}, \omega_\mathrm{m}, \Gamma_\mathrm{m}$
$E, \omega_\mathrm{p}, \kappa$
$\hat{a}, \omega_\mathrm{c}$
g_0

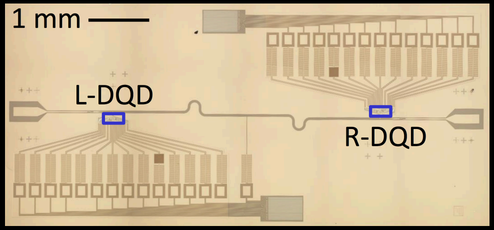
Quantum optics

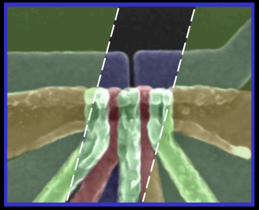


Ion traps



Light matter interaction





Przykłady użycia pakietu QuTiP

Probing many-body dynamics on a 51-atom quantum simulator

Hannes Bernien, Sylvain Schwartz, Alexander Keesling, Harry Levine, Ahmed Omran, Hannes Pichler, Soonwon Choi, Alexander S. Zibrov, Manuel Endres, Markus Greiner, Vladan Vuletić & Mikhail D. Lukin

Nature **551**, 579–584 (2017) | Cite this article

36k Accesses | **978** Citations | **432** Altmetric | Metrics

Unconditional quantum teleportation between distant solid-state quantum bits

W. PFAFF, B. J. HENSEN, H. BERNIEN, S. B. VAN DAM, M. S. BLOK, T. H. TAMINIAU, M. J. TIGGELMAN, R. N. SCHOUTEN, M. MARKHAM, [...] R. HANSON **+2 authors**

Authors Info & Affiliations

SCIENCE • 29 May 2014 • Vol 345, Issue 6196 • pp. 532-535 • DOI: 10.1126/science.1253512

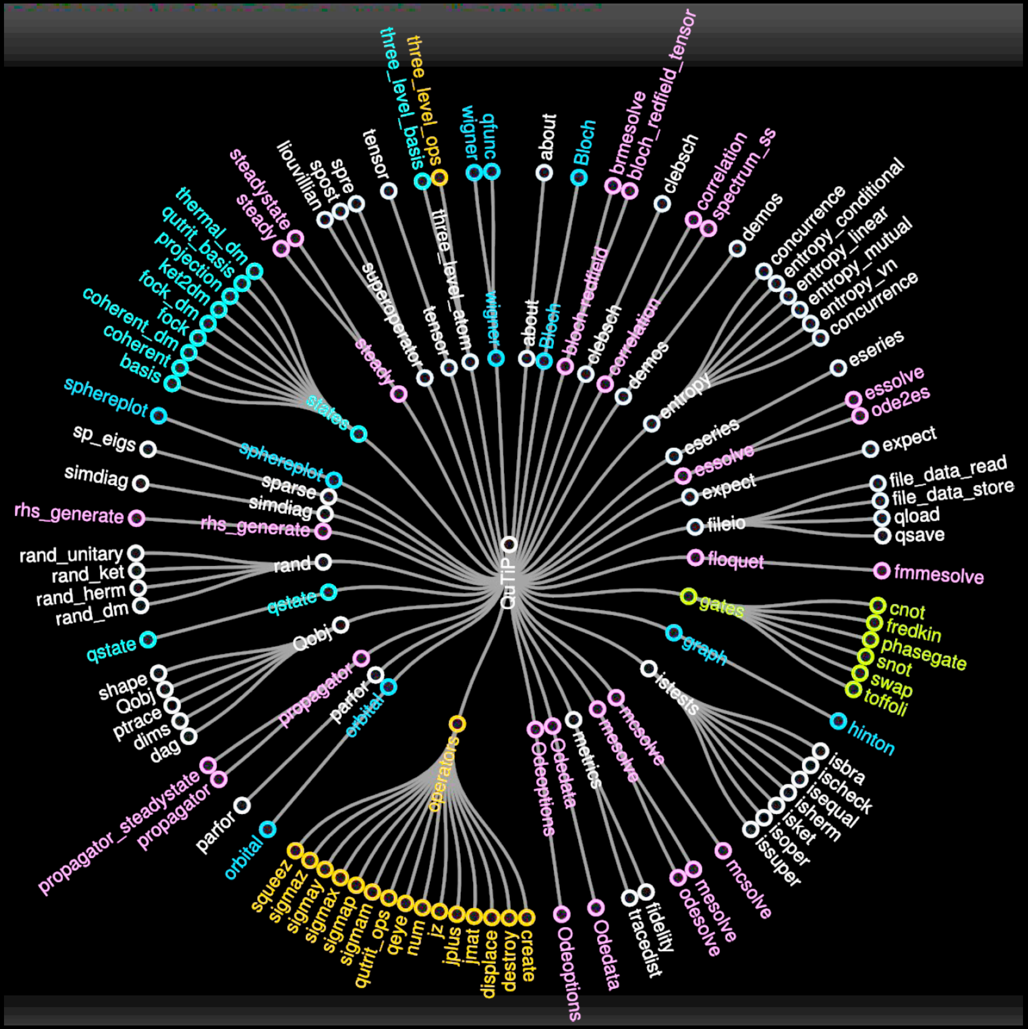
Entanglement-based single-shot detection of a single magnon with a superconducting qubit

DANY LACHANCE-QUIRION, SAMUEL PIOTR WOLSKI, YUTAKA TABUCHI, SHINGO KONO, KOJI USAMI, AND , YASUNOBU NAKAMURA Authors Info & Affiliations

SCIENCE • 24 Jan 2020 • Vol 367, Issue 6476 • pp. 425-428 • DOI: 10.1126/science.aaz9236

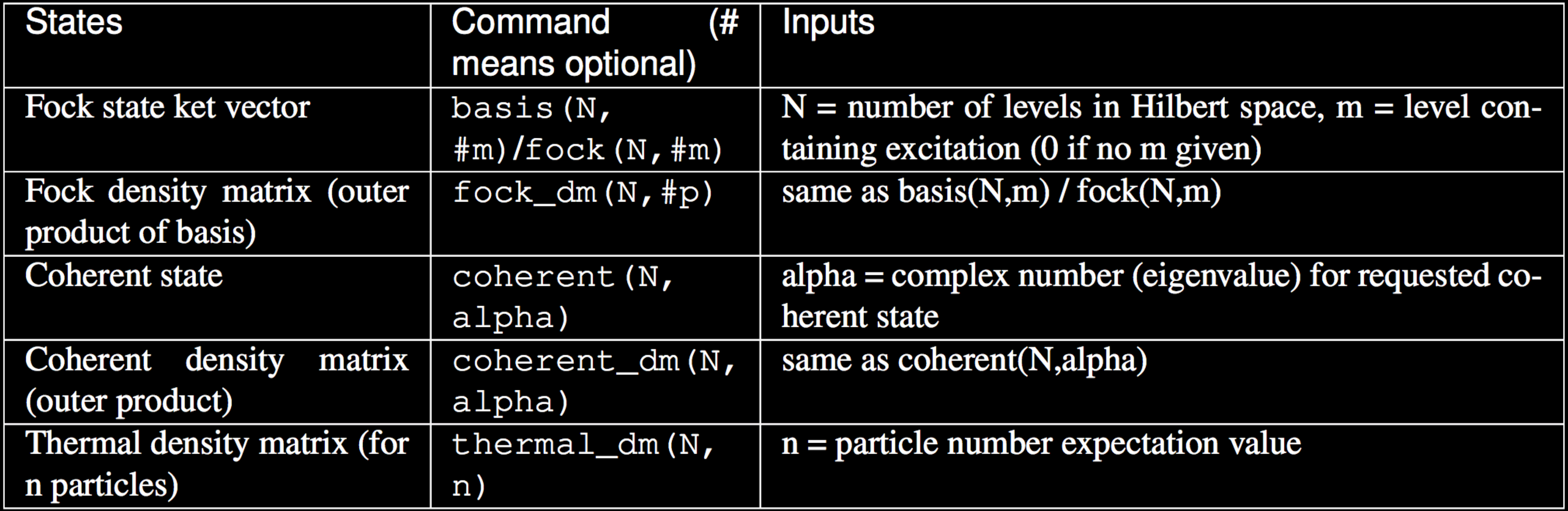
Google NORTHROP GRUMMAN 1QBit Alibaba.com intel Microsoft amazon.com Honeywell NASA IONQ rigetti

Funkcjonalność



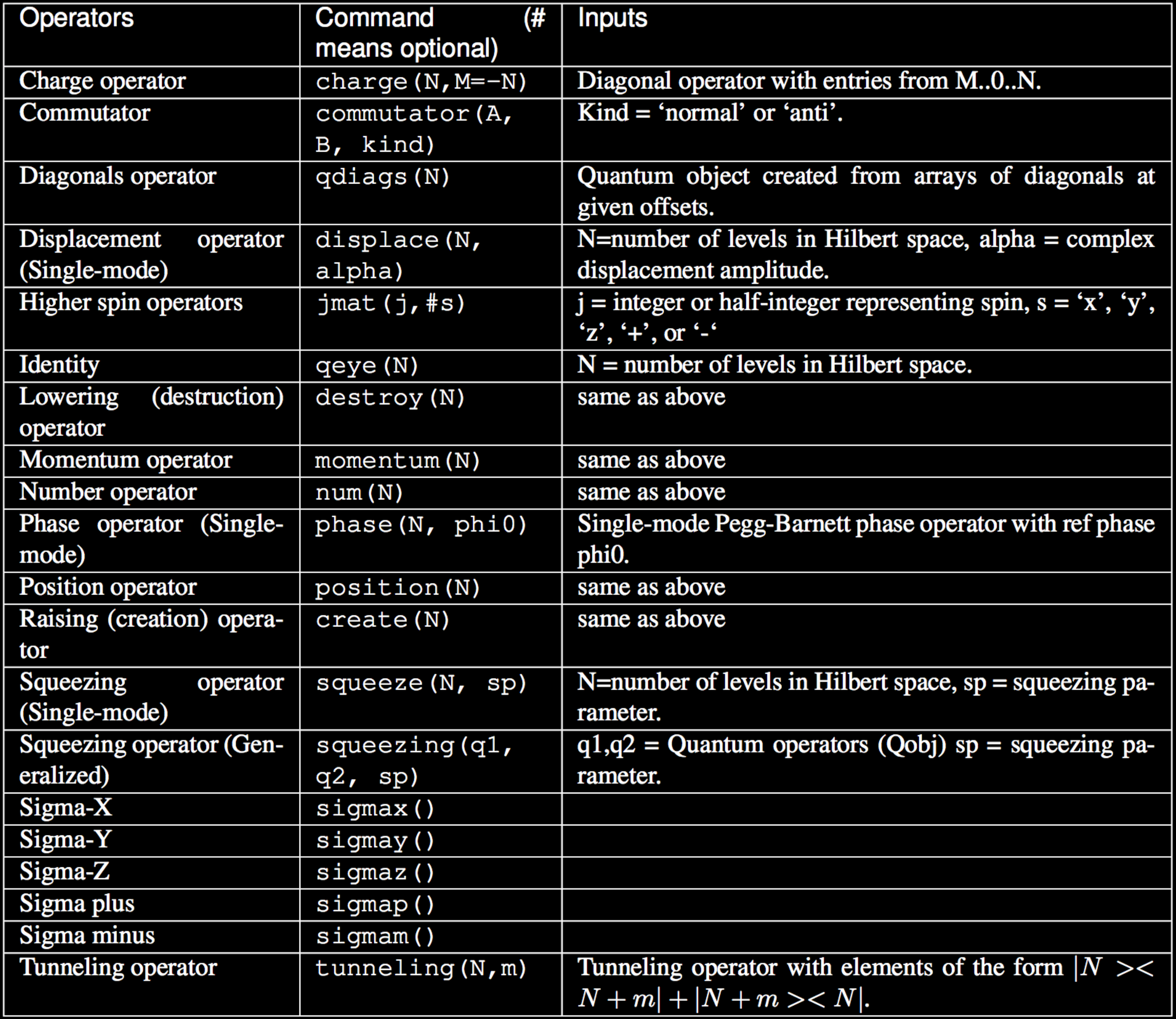
https://qutip.org/downloads/4.7.0/qutip-doc-4.7.pdf

Stany - klasa Qobj



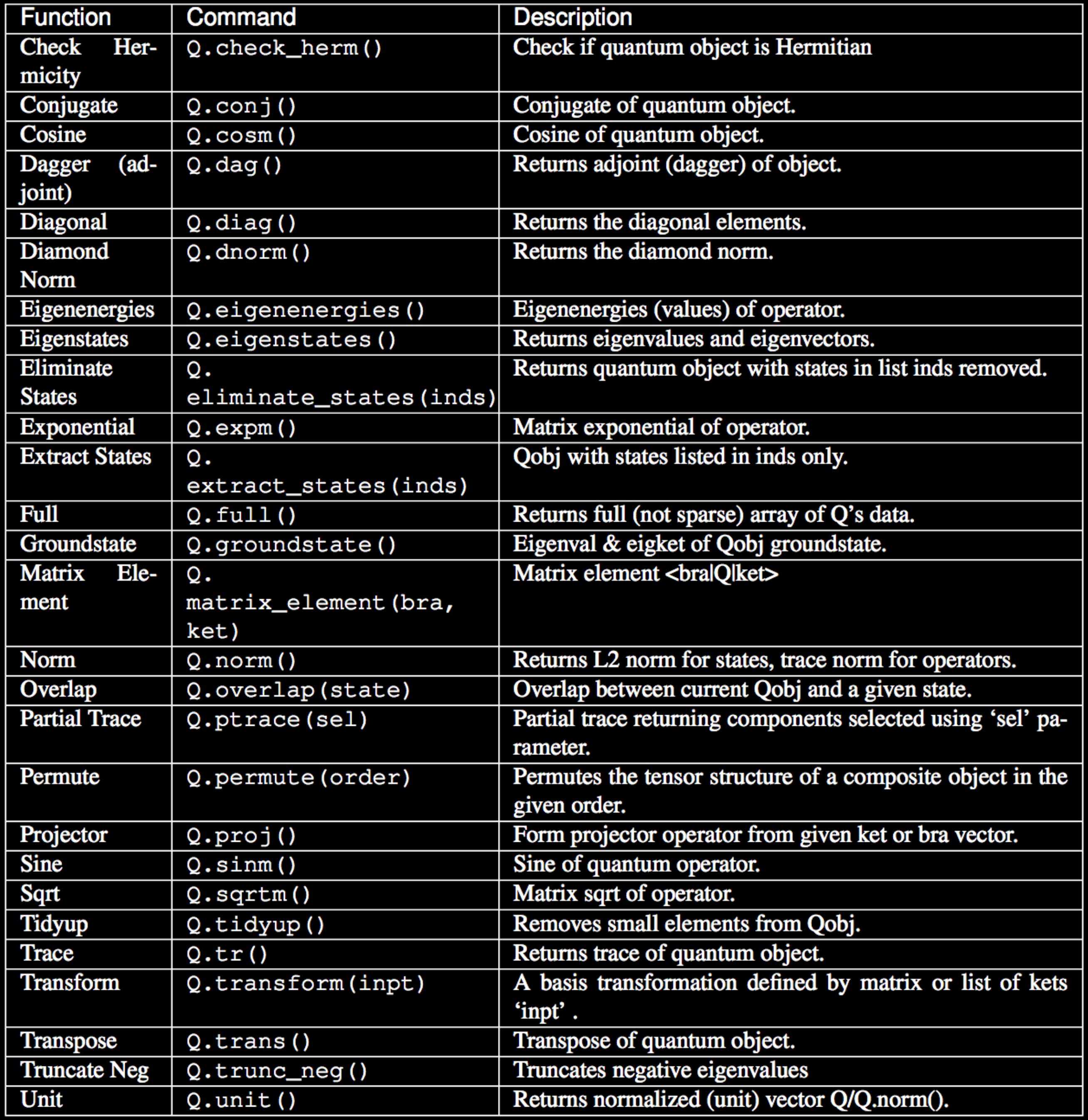
States	Command (# means optional)	Inputs
Fock state ket vector	`basis(N, #m)/fock(N,#m)`	N = number of levels in Hilbert space, m = level containing excitation (0 if no m given)
Fock density matrix (outer product of basis)	`fock_dm(N,#p)`	same as basis(N,m) / fock(N,m)
Coherent state	`coherent(N, alpha)`	alpha = complex number (eigenvalue) for requested coherent state
Coherent density matrix (outer product)	`coherent_dm(N, alpha)`	same as coherent(N,alpha)
Thermal density matrix (for n particles)	`thermal_dm(N, n)`	n = particle number expectation value

Operatory



Operators	Command (# means optional)	Inputs
Charge operator	`charge(N,M=-N)`	Diagonal operator with entries from M..0..N.
Commutator	`commutator(A, B, kind)`	Kind = 'normal' or 'anti'.
Diagonals operator	`qdiags(N)`	Quantum object created from arrays of diagonals at given offsets.
Displacement operator (Single-mode)	`displace(N, alpha)`	N=number of levels in Hilbert space, alpha = complex displacement amplitude.
Higher spin operators	`jmat(j,#s)`	j = integer or half-integer representing spin, s = 'x', 'y', 'z', '+', or '-'
Identity	`qeye(N)`	N = number of levels in Hilbert space.
Lowering (destruction) operator	`destroy(N)`	same as above
Momentum operator	`momentum(N)`	same as above
Number operator	`num(N)`	same as above
Phase operator (Single-mode)	`phase(N, phi0)`	Single-mode Pegg-Barnett phase operator with ref phase phi0.
Position operator	`position(N)`	same as above
Raising (creation) operator	`create(N)`	same as above
Squeezing operator (Single-mode)	`squeeze(N, sp)`	N=number of levels in Hilbert space, sp = squeezing parameter.
Squeezing operator (Generalized)	`squeezing(q1, q2, sp)`	q1,q2 = Quantum operators (Qobj) sp = squeezing parameter.
Sigma-X	`sigmax()`	
Sigma-Y	`sigmay()`	
Sigma-Z	`sigmaz()`	
Sigma plus	`sigmap()`	
Sigma minus	`sigmam()`	
Tunneling operator	`tunneling(N,m)`	Tunneling operator with elements of the form $\|N >< N+m\| + \|N+m >< N\|$.

Funkcje



Function	Command	Description
Check Hermicity	`Q.check_herm()`	Check if quantum object is Hermitian
Conjugate	`Q.conj()`	Conjugate of quantum object.
Cosine	`Q.cosm()`	Cosine of quantum object.
Dagger (adjoint)	`Q.dag()`	Returns adjoint (dagger) of object.
Diagonal	`Q.diag()`	Returns the diagonal elements.
Diamond Norm	`Q.dnorm()`	Returns the diamond norm.
Eigenenergies	`Q.eigenenergies()`	Eigenenergies (values) of operator.
Eigenstates	`Q.eigenstates()`	Returns eigenvalues and eigenvectors.
Eliminate States	`Q.eliminate_states(inds)`	Returns quantum object with states in list inds removed.
Exponential	`Q.expm()`	Matrix exponential of operator.
Extract States	`Q.extract_states(inds)`	Qobj with states listed in inds only.
Full	`Q.full()`	Returns full (not sparse) array of Q's data.
Groundstate	`Q.groundstate()`	Eigenval & eigket of Qobj groundstate.
Matrix Element	`Q.matrix_element(bra, ket)`	Matrix element <bra|Q|ket>
Norm	`Q.norm()`	Returns L2 norm for states, trace norm for operators.
Overlap	`Q.overlap(state)`	Overlap between current Qobj and a given state.
Partial Trace	`Q.ptrace(sel)`	Partial trace returning components selected using 'sel' parameter.
Permute	`Q.permute(order)`	Permutes the tensor structure of a composite object in the given order.
Projector	`Q.proj()`	Form projector operator from given ket or bra vector.
Sine	`Q.sinm()`	Sine of quantum operator.
Sqrt	`Q.sqrtm()`	Matrix sqrt of operator.
Tidyup	`Q.tidyup()`	Removes small elements from Qobj.
Trace	`Q.tr()`	Returns trace of quantum object.
Transform	`Q.transform(inpt)`	A basis transformation defined by matrix or list of kets 'inpt' .
Transpose	`Q.trans()`	Transpose of quantum object.
Truncate Neg	`Q.trunc_neg()`	Truncates negative eigenvalues
Unit	`Q.unit()`	Returns normalized (unit) vector Q/Q.norm().

Ewolucja czasowa wektorów stanu i macierzy gęstości - **solvers**

def mesolve(H, rho0, tlist, c_ops=None, e_ops=None, args=None, options=None, progress_bar=None, _safe_mode=True):

def mcsolve(H, psi0, tlist, c_ops=[], e_ops=[], ntraj=0, args={}, options=None, progress_bar=True,
map_func=parallel_map, map_kwargs={}, _safe_mode=True):

def sesolve(H, psi0, tlist, e_ops=None, args=None, options=None, progress_bar=None, _safe_mode=True):

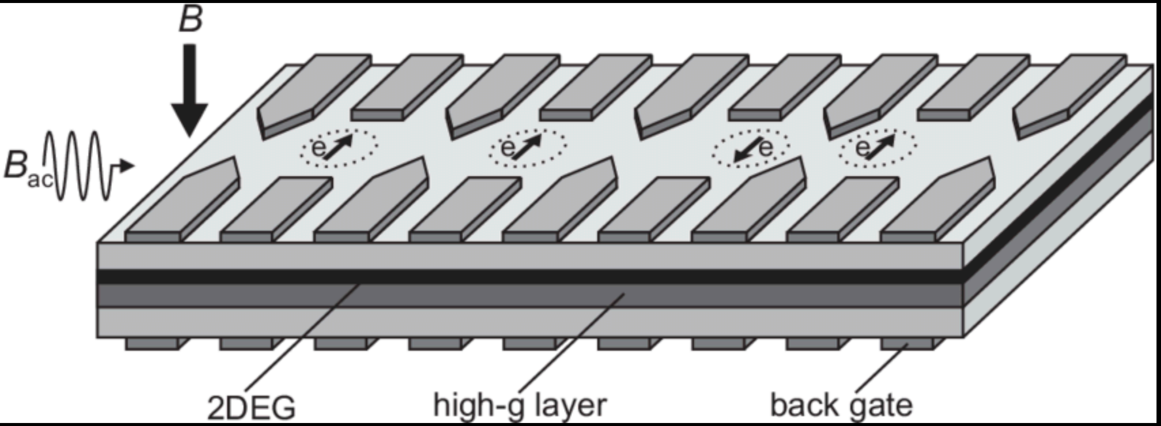
def brmesolve(H, psi0, tlist, a_ops=[], e_ops=[], c_ops=[], args={}, use_secular=True, sec_cutoff = 0.1, tol=qset.atol, spectra_cb=None, options=None, progress_bar=None, _safe_mode=True, verbose=False):

def ssesolve(H, psi0, times, sc_ops=[], e_ops=[], _safe_mode=True, args={}, **kwargs):

https://qutip.org/docs/latest/modules/index.html

I wiecej…

Spin w kropkach kwantowych



Daniel Loss DP DiVincenzo

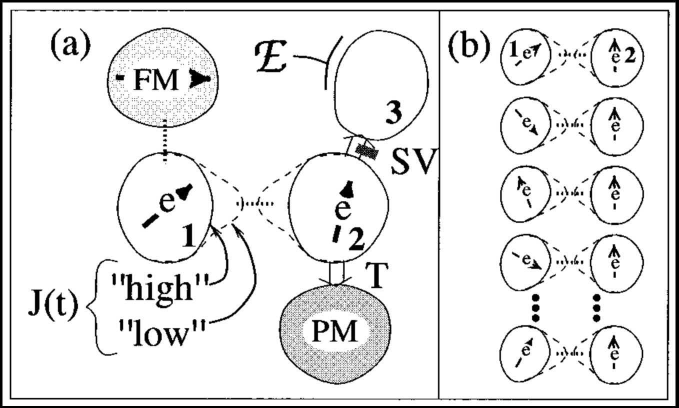
$$H = \sum_{\langle ij \rangle} J_{ij}(t) S_i \cdot S_j + \sum_i (g_i \mu_B B_i)(t) \cdot S_i$$

$$|\Psi\rangle = \alpha|\uparrow\rangle + \beta|\downarrow\rangle$$

$$|1\rangle = |\downarrow\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

$$S_i = \frac{\hbar}{2}\sigma_i$$

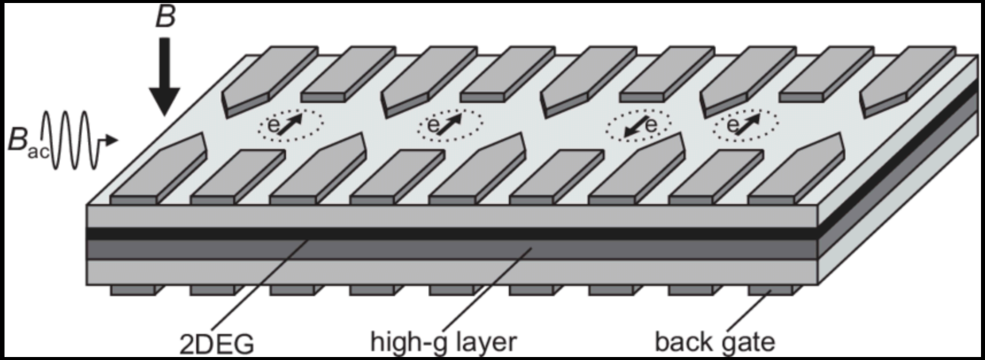
$$|0\rangle = |\uparrow\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

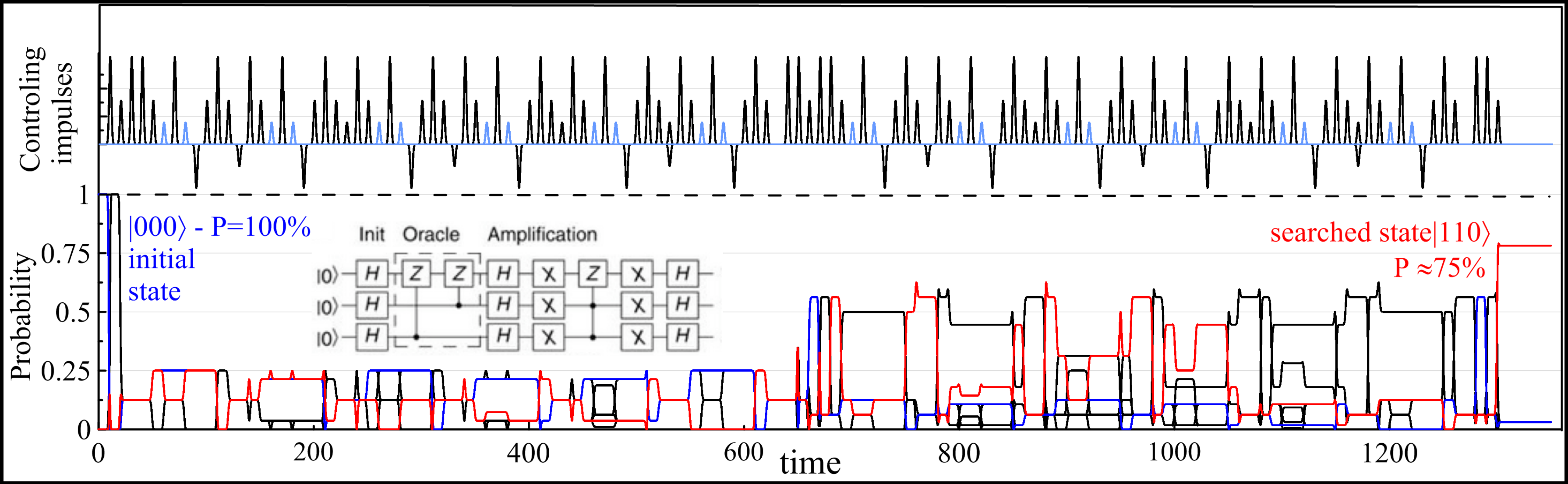


Quantum computation with quantum dots
D Loss, DP DiVincenzo PRA **57** (1), 120 (1997)

Symulacja algorytmu Grovera

$$\begin{aligned} H(t) = &\vec{B}_1(t) \cdot \vec{\sigma_1} \otimes I \otimes I \\ &+\vec{B}_2(t) \cdot I \otimes \vec{\sigma_2} \otimes I \\ &+\vec{B}_3(t) \cdot I \otimes I \otimes \vec{\sigma_3} \\ &+J_{12}(t) \cdot \vec{\sigma_1} \otimes \vec{\sigma_2} \otimes I \\ &+J_{23}(t) \cdot I \otimes I \otimes \vec{\sigma_2} \otimes \vec{\sigma_3} \end{aligned}$$

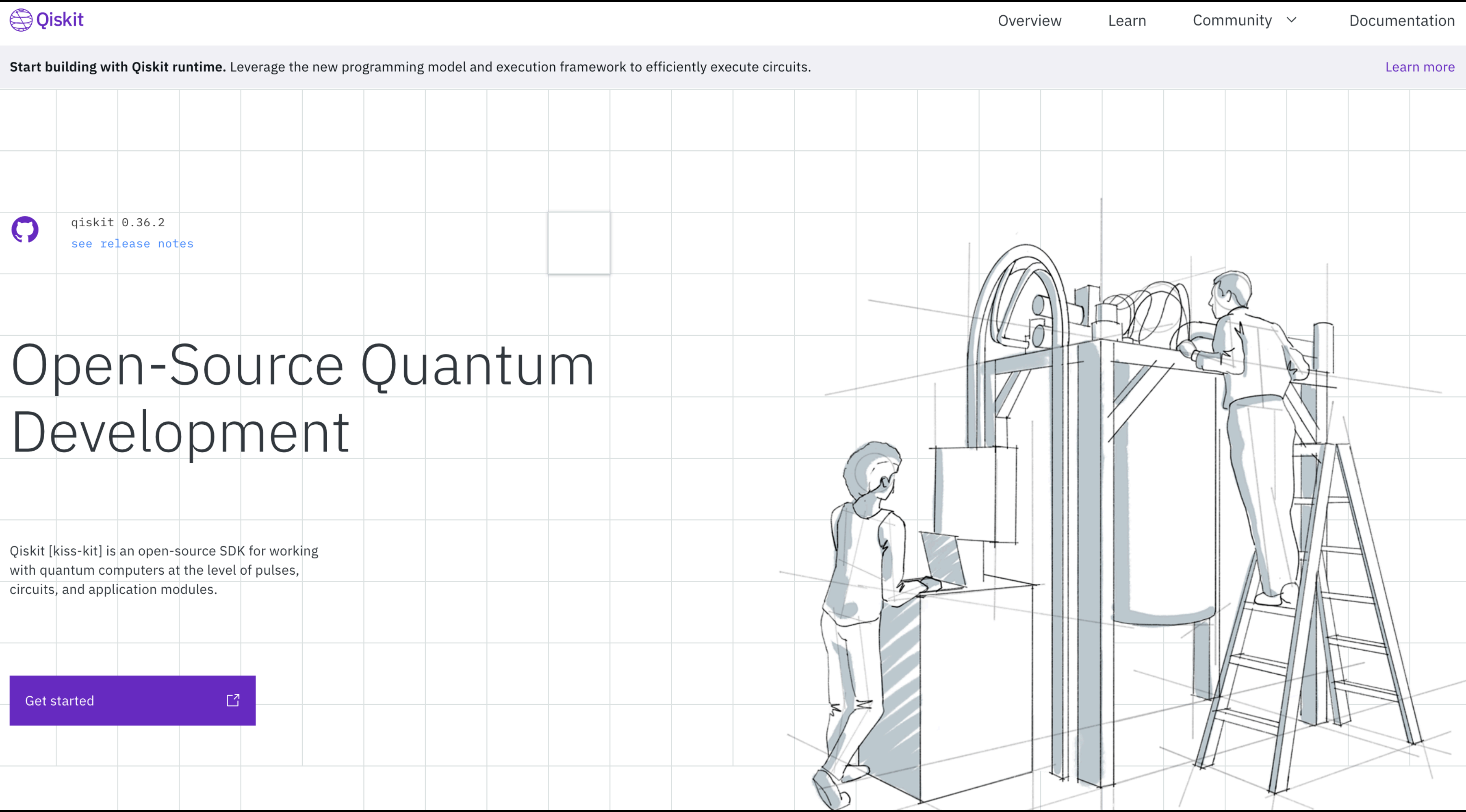




Obliczenia kwantowe -Oprogramowanie

- https://github.com/qosf/awesome-quantum-software
- https://quantumcomputingreport.com/tools/
- https://github.com/desireevl/awesome-quantum-computing

Qiskit (IBM) - https://qiskit.org/



Qiskit

Overview Learn Community Documentation

Start building with Qiskit runtime. Leverage the new programming model and execution framework to efficiently execute circuits. Learn more

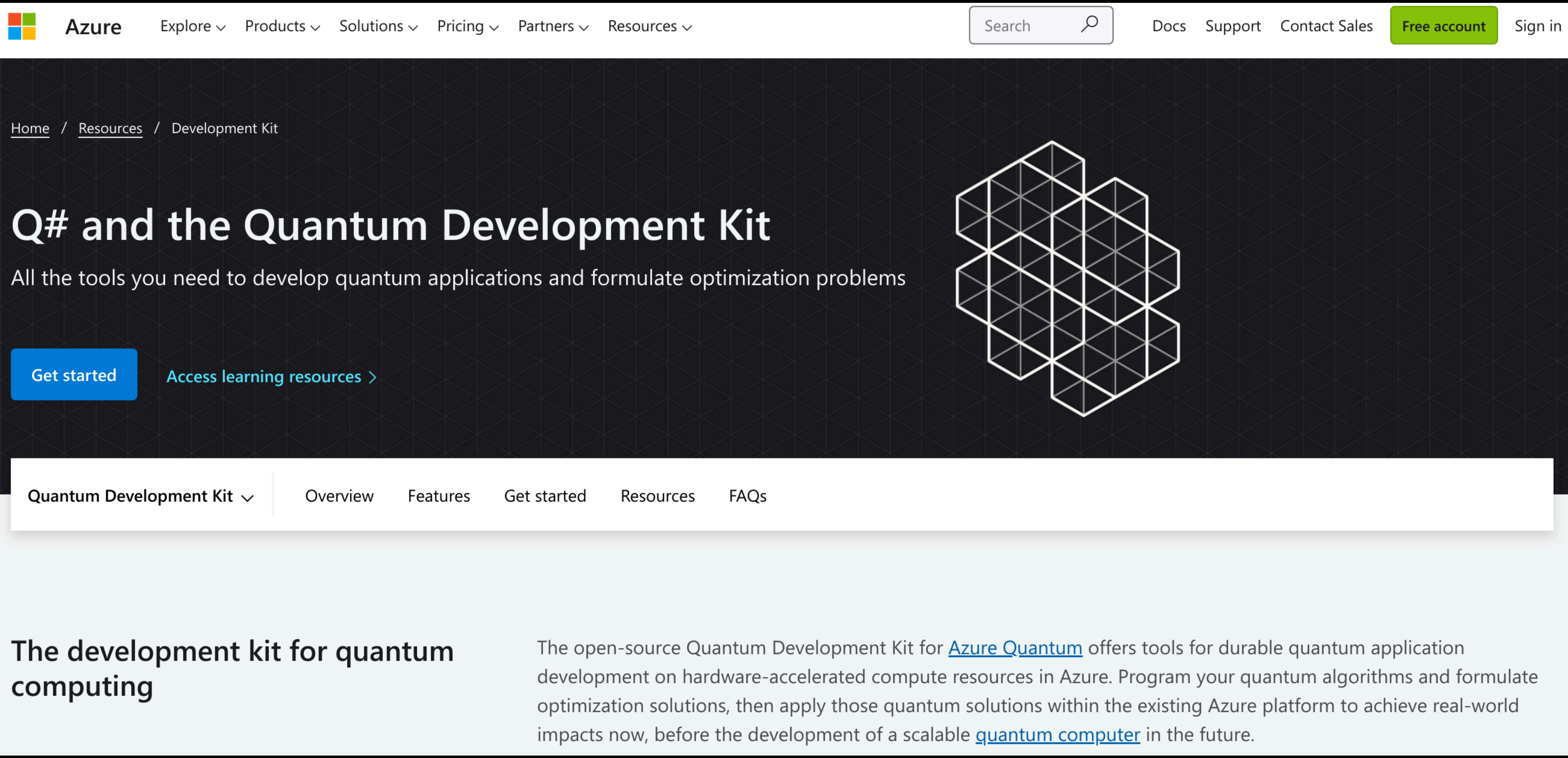
qiskit 0.36.2
see release notes

Open-Source Quantum Development

Qiskit [kiss-kit] is an open-source SDK for working with quantum computers at the level of pulses, circuits, and application modules.

Get started

Q #(Microsoft) - https://azure.microsoft.com/en-us/resources/development-kit/quantum-computing/



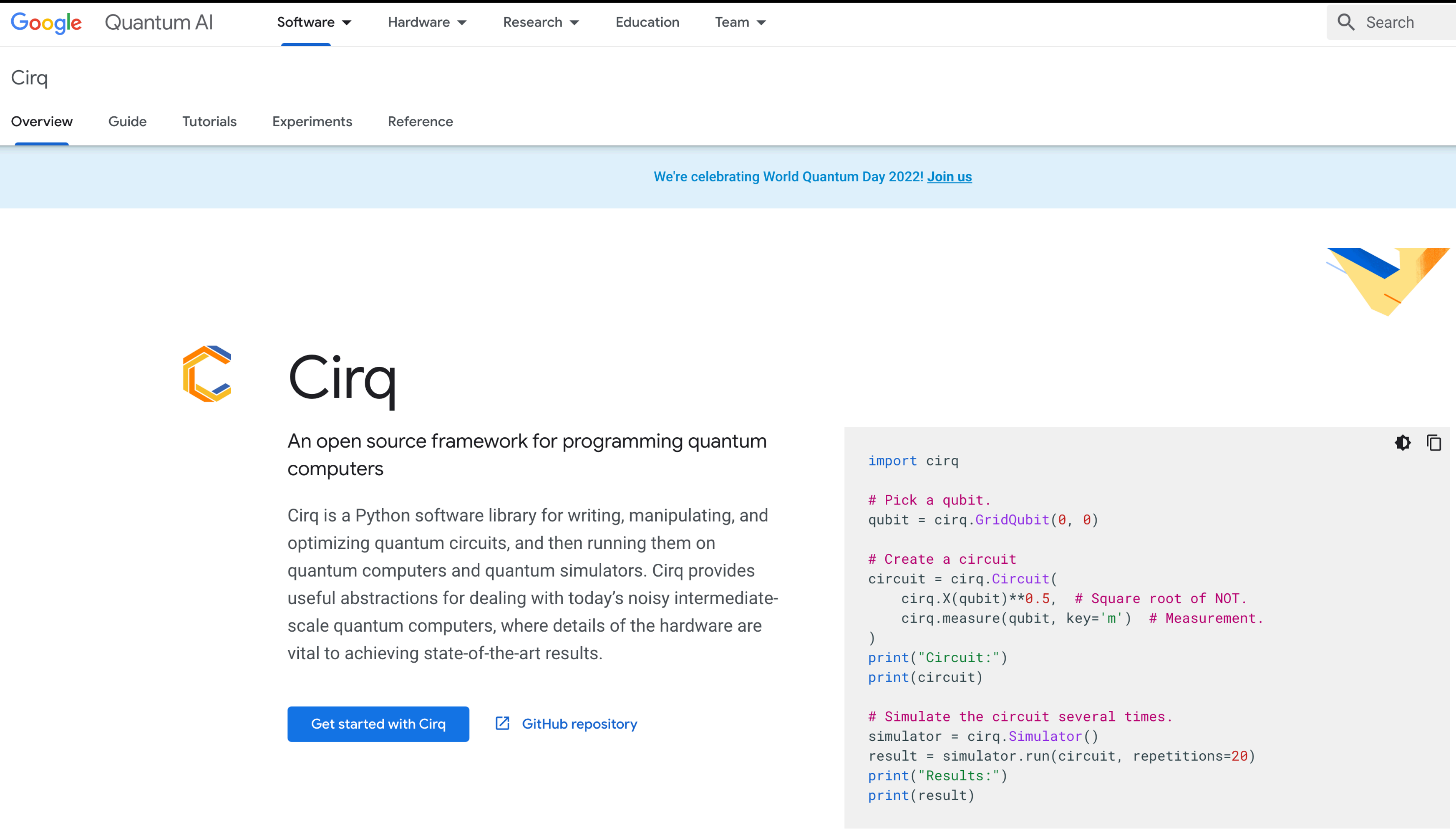
Q# and the Quantum Development Kit

All the tools you need to develop quantum applications and formulate optimization problems

The development kit for quantum computing

The open-source Quantum Development Kit for Azure Quantum offers tools for durable quantum application development on hardware-accelerated compute resources in Azure. Program your quantum algorithms and formulate optimization solutions, then apply those quantum solutions within the existing Azure platform to achieve real-world impacts now, before the development of a scalable quantum computer in the future.

Cirq (Google) -https://quantumai.google/cirq



We're celebrating World Quantum Day 2022! Join us

Cirq

An open source framework for programming quantum computers

Cirq is a Python software library for writing, manipulating, and optimizing quantum circuits, and then running them on quantum computers and quantum simulators. Cirq provides useful abstractions for dealing with today's noisy intermediate-scale quantum computers, where details of the hardware are vital to achieving state-of-the-art results.

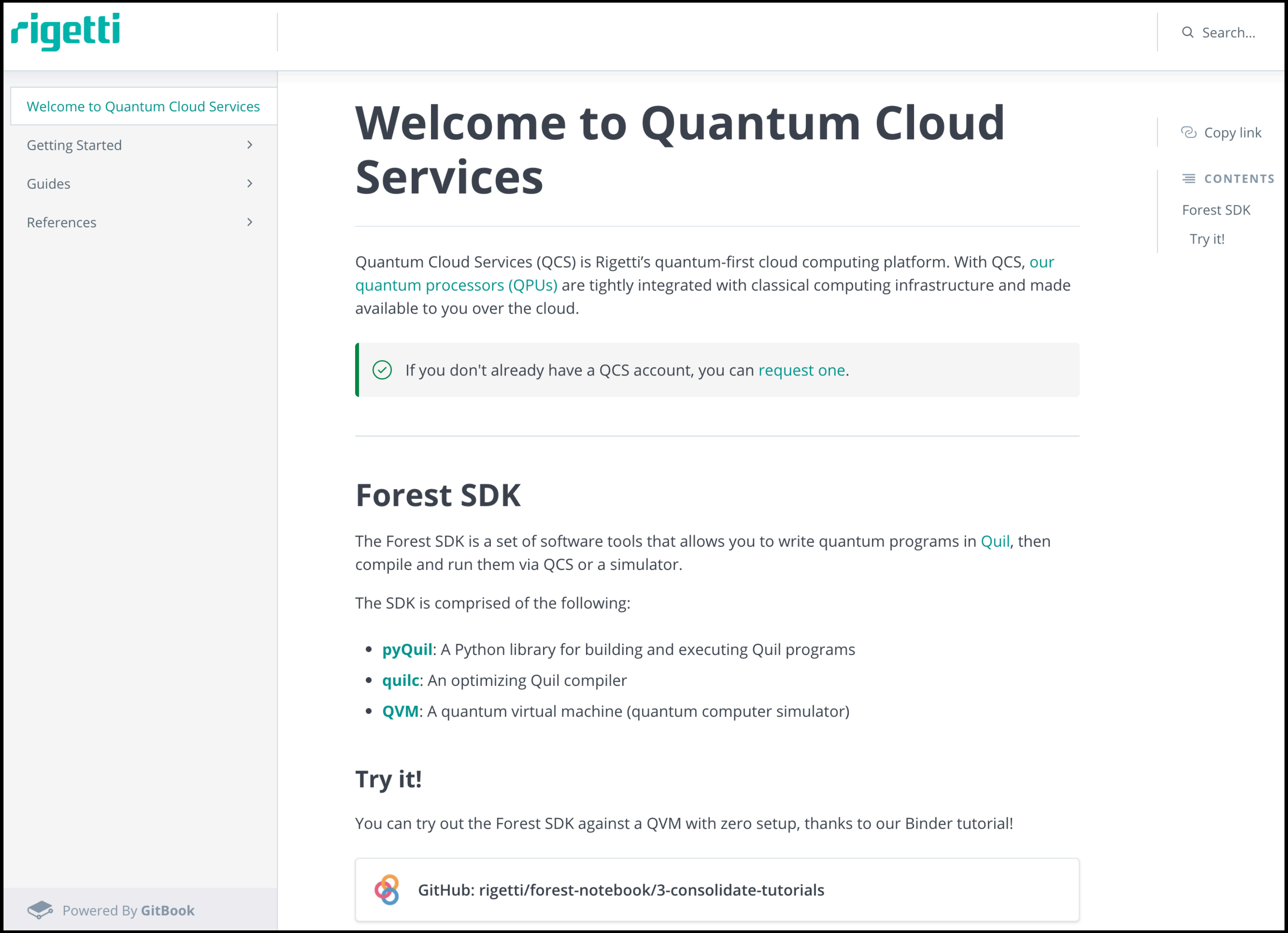
```python
import cirq

# Pick a qubit.
qubit = cirq.GridQubit(0, 0)

# Create a circuit
circuit = cirq.Circuit(
    cirq.X(qubit)**0.5,  # Square root of NOT.
    cirq.measure(qubit, key='m')  # Measurement.
)
print("Circuit:")
print(circuit)

# Simulate the circuit several times.
simulator = cirq.Simulator()
result = simulator.run(circuit, repetitions=20)
print("Results:")
print(result)
```

Forest (Rigetti)- https://docs.rigetti.com/qcs/



Welcome to Quantum Cloud Services

Quantum Cloud Services (QCS) is Rigetti's quantum-first cloud computing platform. With QCS, our quantum processors (QPUs) are tightly integrated with classical computing infrastructure and made available to you over the cloud.

If you don't already have a QCS account, you can request one.

Forest SDK

The Forest SDK is a set of software tools that allows you to write quantum programs in Quil, then compile and run them via QCS or a simulator.

The SDK is comprised of the following:

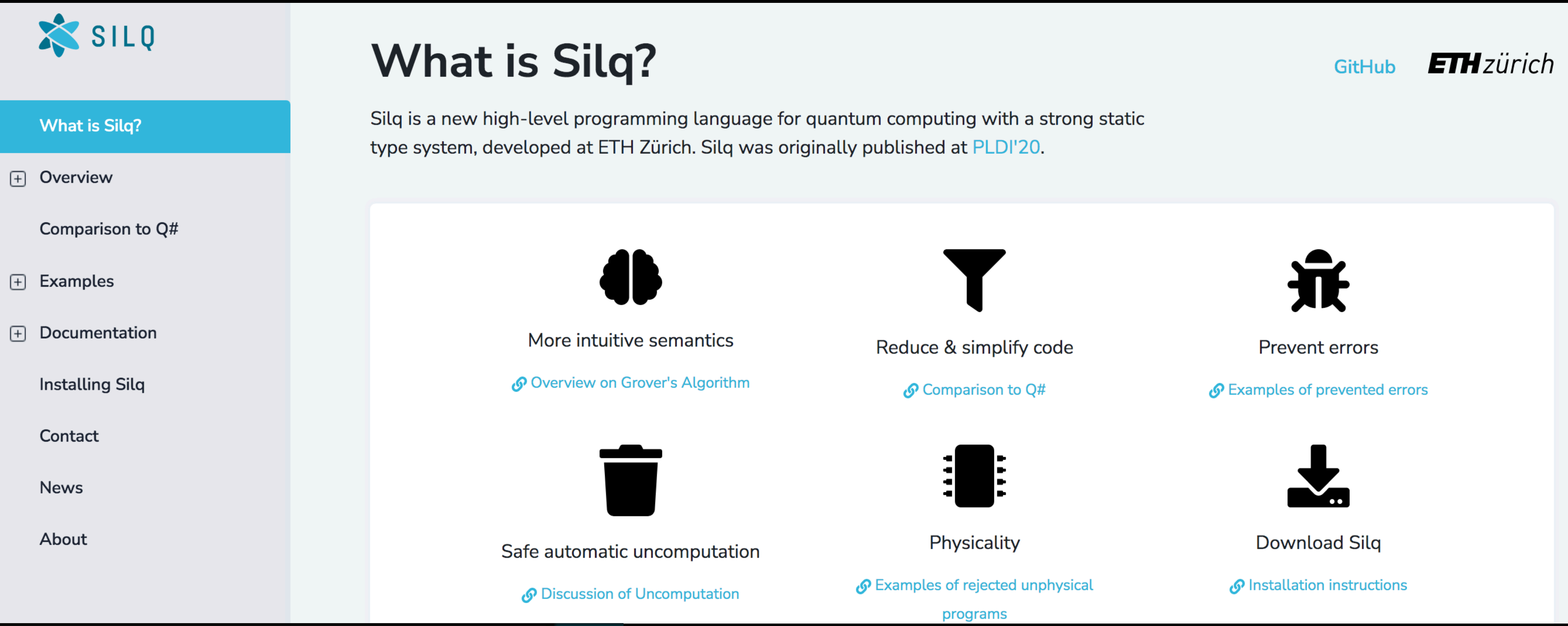
- **pyQuil**: A Python library for building and executing Quil programs
- **quilc**: An optimizing Quil compiler
- **QVM**: A quantum virtual machine (quantum computer simulator)

Try it!

You can try out the Forest SDK against a QVM with zero setup, thanks to our Binder tutorial!

GitHub: rigetti/forest-notebook/3-consolidate-tutorials

Sliq (ETH Zurich) - https://silq.ethz.ch/



SILQ

- What is Silq?
- Overview
- Comparison to Q#
- Examples
- Documentation
- Installing Silq
- Contact
- News
- About

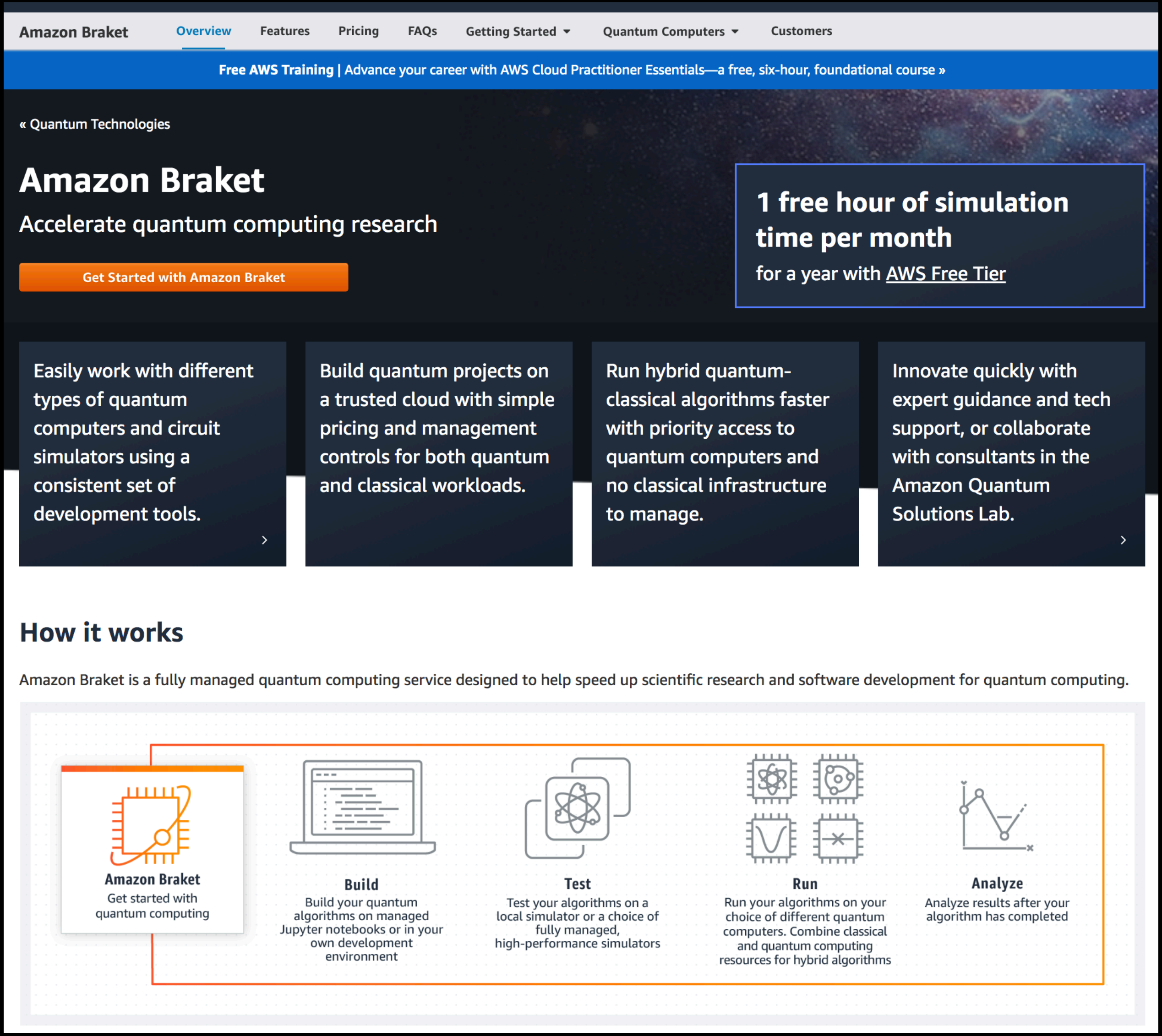
What is Silq?

GitHub ETH zürich

Silq is a new high-level programming language for quantum computing with a strong static type system, developed at ETH Zürich. Silq was originally published at PLDI'20.

More intuitive semantics	Reduce & simplify code	Prevent errors
Overview on Grover's Algorithm	Comparison to Q#	Examples of prevented errors
Safe automatic uncomputation	Physicality	Download Silq
Discussion of Uncomputation	Examples of rejected unphysical programs	Installation instructions

Amazon Braket https://aws.amazon.com/braket/



Free AWS Training | Advance your career with AWS Cloud Practitioner Essentials—a free, six-hour, foundational course »

« Quantum Technologies

Amazon Braket

Accelerate quantum computing research

1 free hour of simulation time per month
for a year with AWS Free Tier

Easily work with different types of quantum computers and circuit simulators using a consistent set of development tools.

Build quantum projects on a trusted cloud with simple pricing and management controls for both quantum and classical workloads.

Run hybrid quantum-classical algorithms faster with priority access to quantum computers and no classical infrastructure to manage.

Innovate quickly with expert guidance and tech support, or collaborate with consultants in the Amazon Quantum Solutions Lab.

How it works

Amazon Braket is a fully managed quantum computing service designed to help speed up scientific research and software development for quantum computing.

Amazon Braket
Get started with quantum computing

Build
Build your quantum algorithms on managed Jupyter notebooks or in your own development environment

Test
Test your algorithms on a local simulator or a choice of fully managed, high-performance simulators

Run
Run your algorithms on your choice of different quantum computers. Combine classical and quantum computing resources for hybrid algorithms

Analyze
Analyze results after your algorithm has completed

Dziękuję za uwagę

Fundusze Europejskie
Inteligentny Rozwój

Rzeczpospolita Polska

FNP Fundacja na rzecz Nauki Polskiej

Unia Europejska
Europejski Fundusz Rozwoju Regionalnego

Financial support within POIR.04.04.00-00-5CE6/18 project carried out within the HOMING programme of the Foundation for Polish Science co-financed by the European Union under the European Regional Development Fund