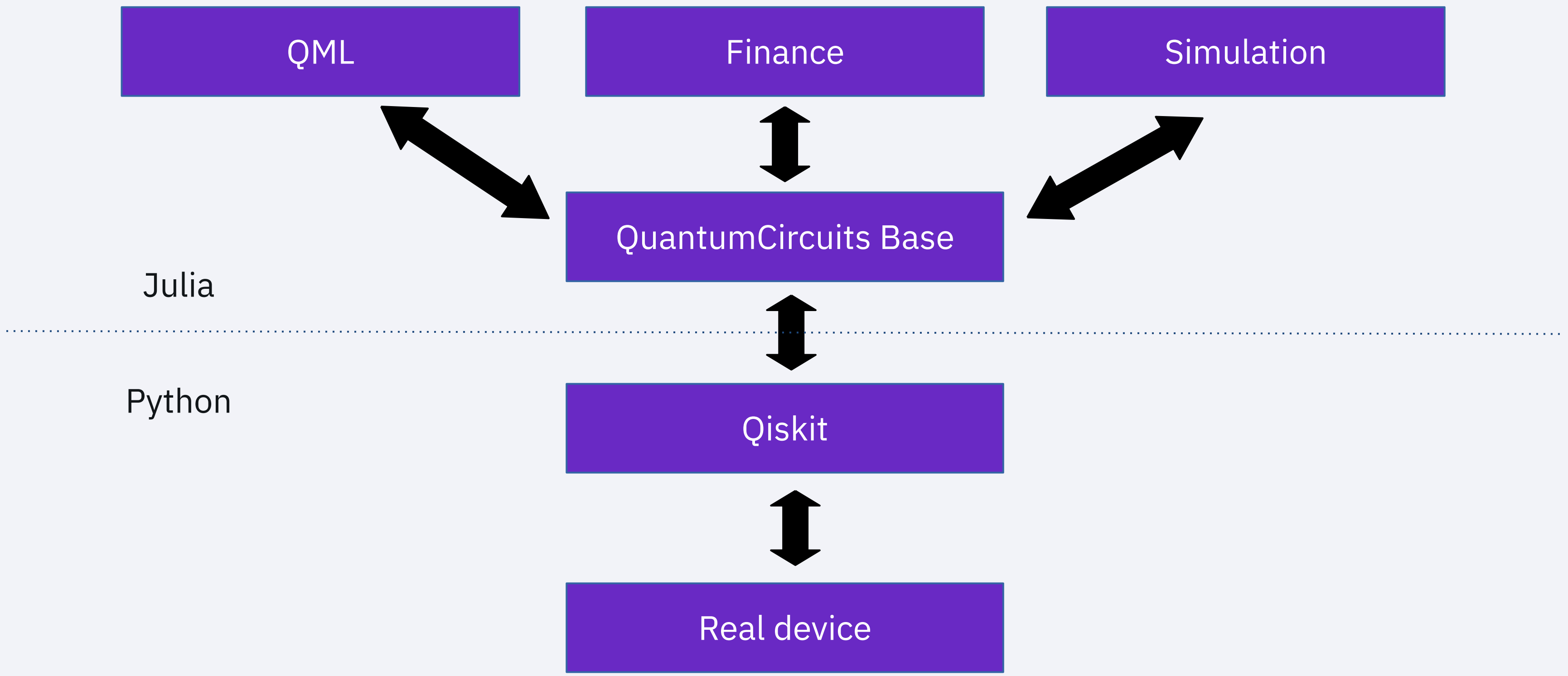
#16 Julia in Qiskit

Sahar Ben Rached, Rafał Pracht

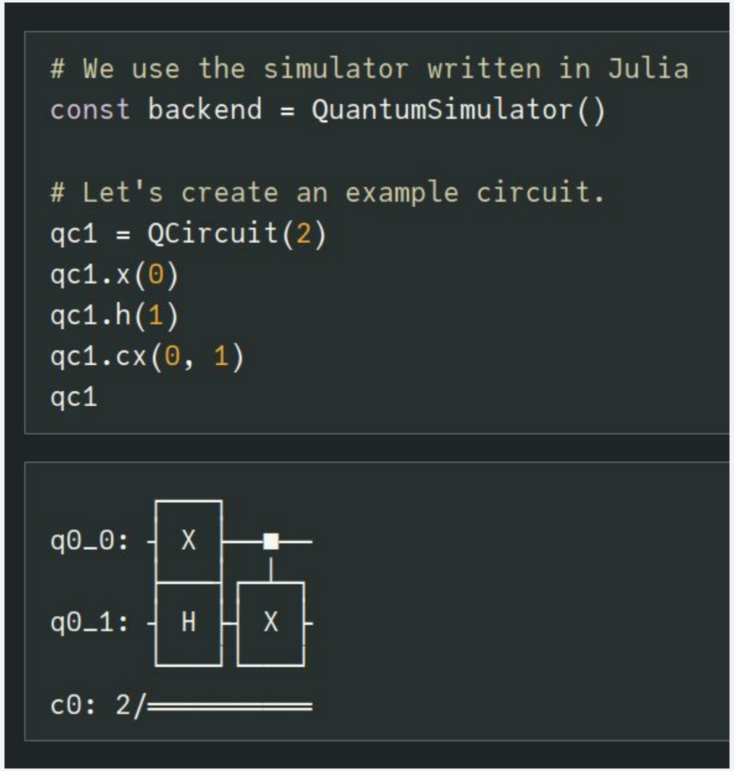
Mentored by John Lapeyre and Jim Garrison

QuantumCircuits library

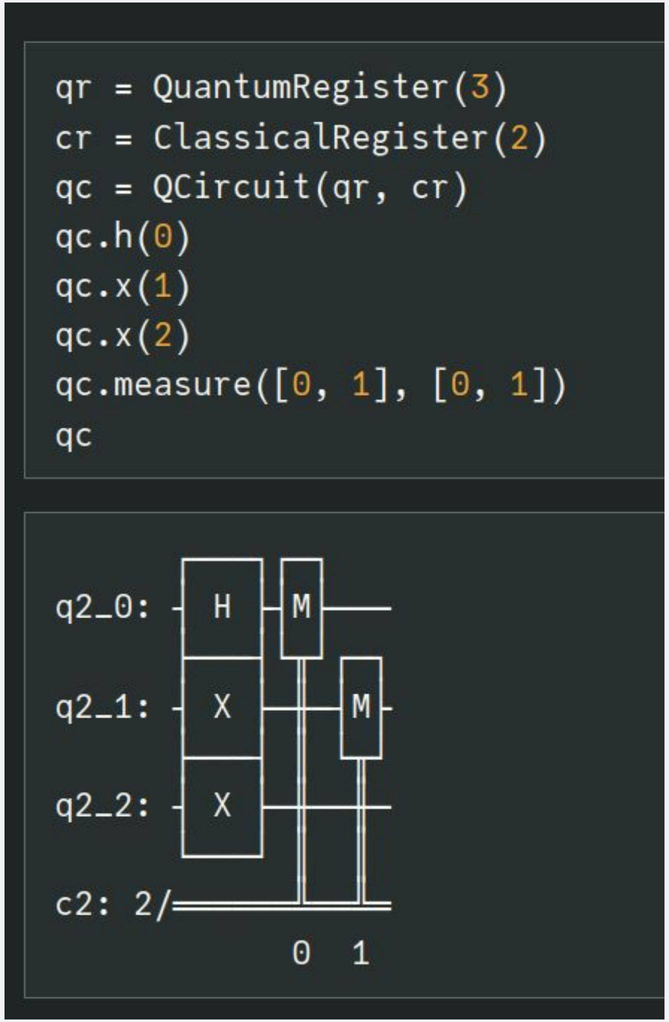
QuantumCircuits Architecture



Qiskit



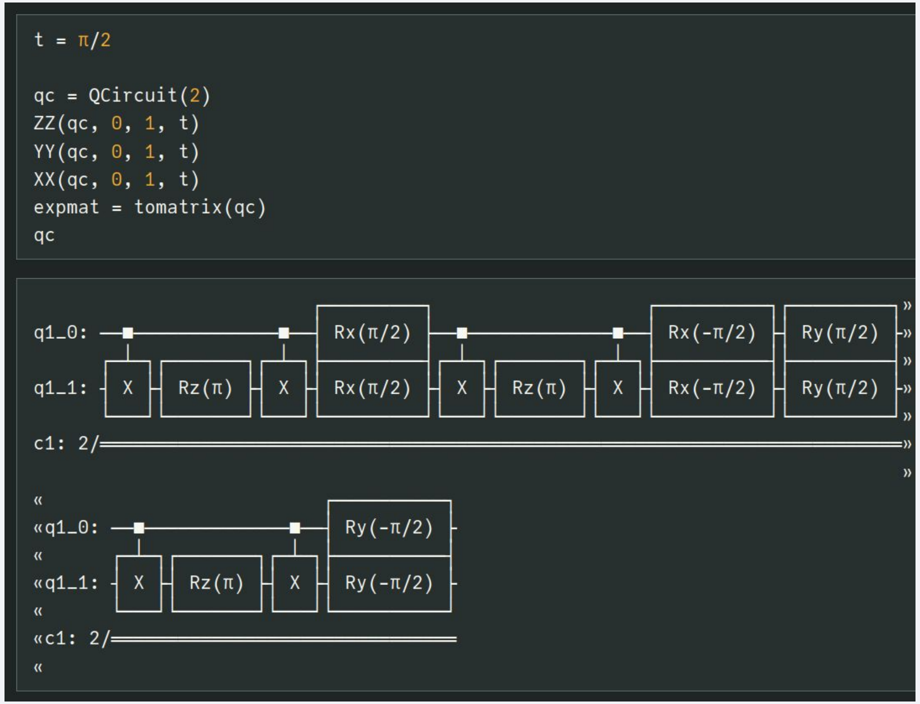
```
# We use the simulator written in Julia
const backend = QuantumSimulator()

# Let's create an example circuit.
qc1 = QCircuit(2)
qc1.x(0)
qc1.h(1)
qc1.cx(0, 1)
qc1
```



```
qr = QuantumRegister(3)
cr = ClassicalRegister(2)
qc = QCircuit(qr, cr)
qc.h(0)
qc.x(1)
qc.x(2)
qc.measure([0, 1], [0, 1])
qc
```

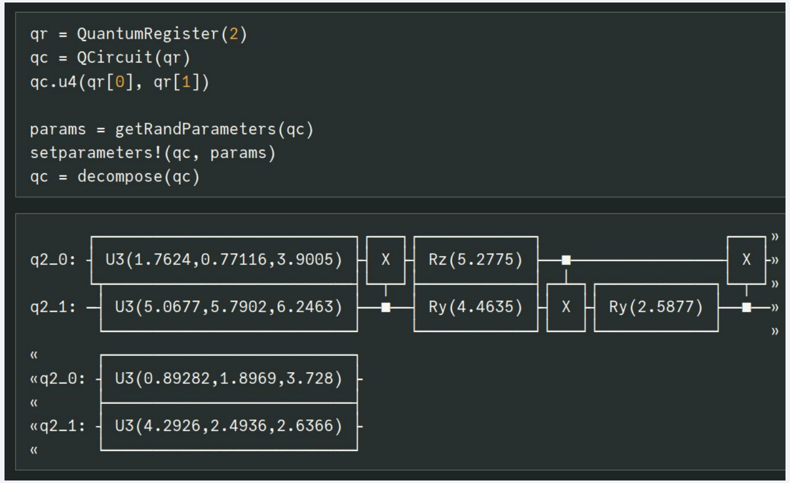
see: https://adgnitio.github.io/QuantumCircuits.jl/dev/

Cartan's KAK decomposition



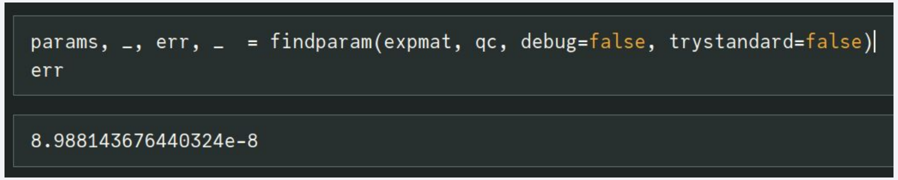
```
t = π/2

qc = QCircuit(2)
ZZ(qc, 0, 1, t)
YY(qc, 0, 1, t)
XX(qc, 0, 1, t)
expmat = tomatrix(qc)
qc
```

```
                         ┌─────────┐                   ┌──────────┐┌─────────┐»
q1_0: ──■─────────────■──┤ Rx(π/2) ├──■─────────────■──┤ Rx(-π/2) ├┤ Ry(π/2) ├»
      ┌─┴─┐┌───────┐┌─┴─┐├─────────┤┌─┴─┐┌───────┐┌─┴─┐├──────────┤├─────────┤»
q1_1: ┤ X ├┤ Rz(π) ├┤ X ├┤ Rx(π/2) ├┤ X ├┤ Rz(π) ├┤ X ├┤ Rx(-π/2) ├┤ Ry(π/2) ├»
      └───┘└───────┘└───┘└─────────┘└───┘└───────┘└───┘└──────────┘└─────────┘»
c1: 2/════════════════════════════════════════════════════════════════════════»
                                                                              »
«                         ┌──────────┐
«q1_0: ──■─────────────■──┤ Ry(-π/2) ├
«      ┌─┴─┐┌───────┐┌─┴─┐├──────────┤
«q1_1: ┤ X ├┤ Rz(π) ├┤ X ├┤ Ry(-π/2) ├
«      └───┘└───────┘└───┘└──────────┘
«c1: 2/═══════════════════════════════
«
```



```
qr = QuantumRegister(2)
qc = QCircuit(qr)
qc.u4(qr[0], qr[1])

params = getRandParameters(qc)
setparameters!(qc, params)
qc = decompose(qc)
```

```
      ┌─────────────────────────────┐┌───┐┌────────────┐                   ┌───┐»
q2_0: ┤ U3(1.7624,0.77116,3.9005)   ├┤ X ├┤ Rz(5.2775) ├──■────────────────┤ X ├»
      ├─────────────────────────────┤└─┬─┘├────────────┤┌─┴─┐┌────────────┐└─┬─┘»
q2_1: ┤ U3(5.0677,5.7902,6.2463)    ├──■──┤ Ry(4.4635) ├┤ X ├┤ Ry(2.5877) ├──■──»
      └─────────────────────────────┘     └────────────┘└───┘└────────────┘     »
«      ┌─────────────────────────────┐
«q2_0: ┤ U3(0.89282,1.8969,3.728)    ├
«      ├─────────────────────────────┤
«q2_1: ┤ U3(4.2926,2.4936,2.6366)    ├
«      └─────────────────────────────┘
```



```
params, _, err, _  = findparam(expmat, qc, debug=false, trystandard=false)
err
```

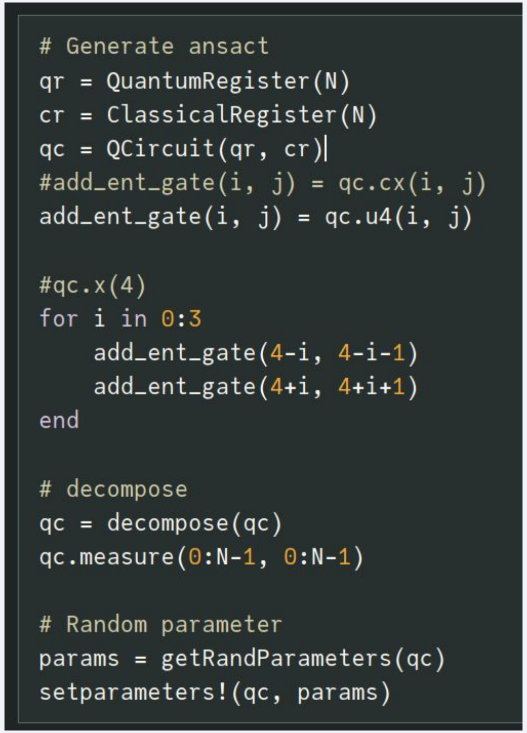
```
8.988143676440324e-8
```

see: https://adgnitio.github.io/QuantumCircuits.jl/dev/Simulation/U4_Cartan_decomposition/

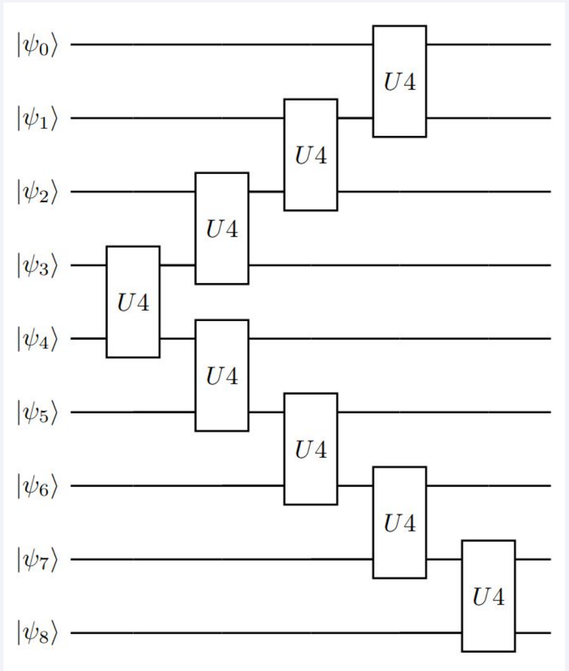
Log-Normal state preparation

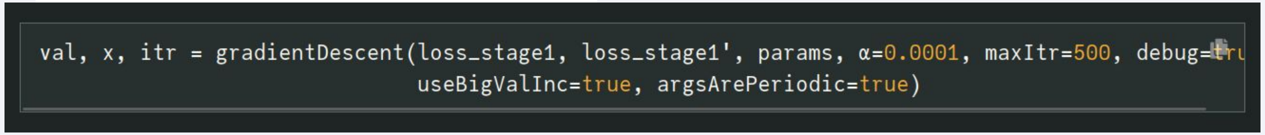
Qiskit

The preparing a quantum state in general, an exponential number of gates, O(2^n), are needed



```
# Generate ansact
qr = QuantumRegister(N)
cr = ClassicalRegister(N)
qc = QCircuit(qr, cr)
#add_ent_gate(i, j) = qc.cx(i, j)
add_ent_gate(i, j) = qc.u4(i, j)

#qc.x(4)
for i in 0:3
    add_ent_gate(4-i, 4-i-1)
    add_ent_gate(4+i, 4+i+1)
end

# decompose
qc = decompose(qc)
qc.measure(0:N-1, 0:N-1)

# Random parameter
params = getRandParameters(qc)
setparameters!(qc, params)
```

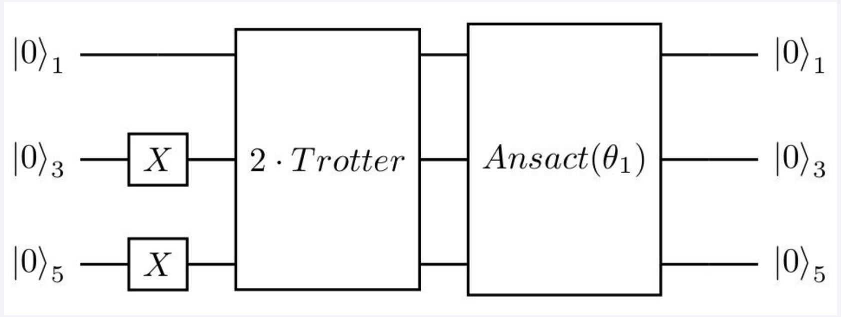


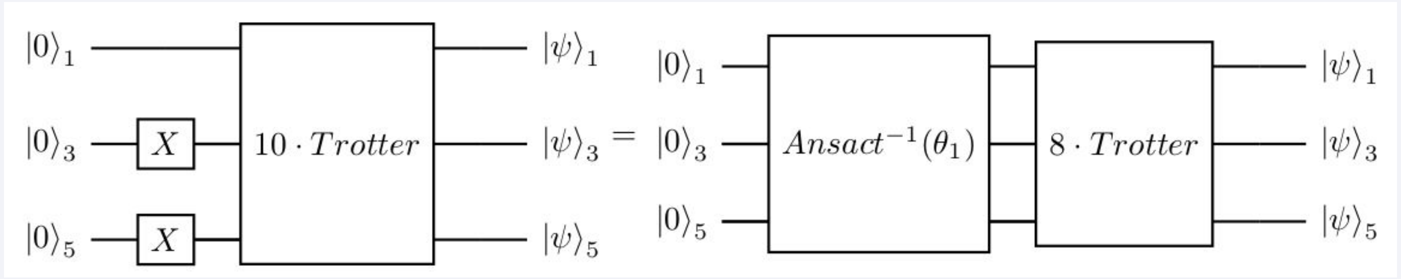


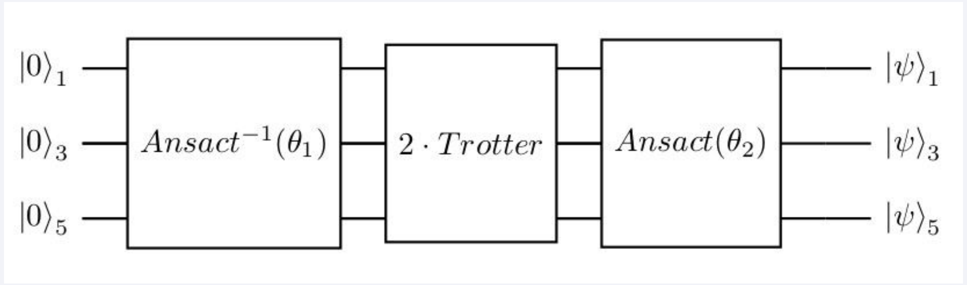
```
val, x, itr = gradientDescent(loss_stage1, loss_stage1', params, α=0.0001, maxItr=500, debug=tru
                                          useBigValInc=true, argsArePeriodic=true)
```

see: https://adgnitio.github.io/QuantumCircuits.jl/dev/examples/state_preparation/

Trotterization



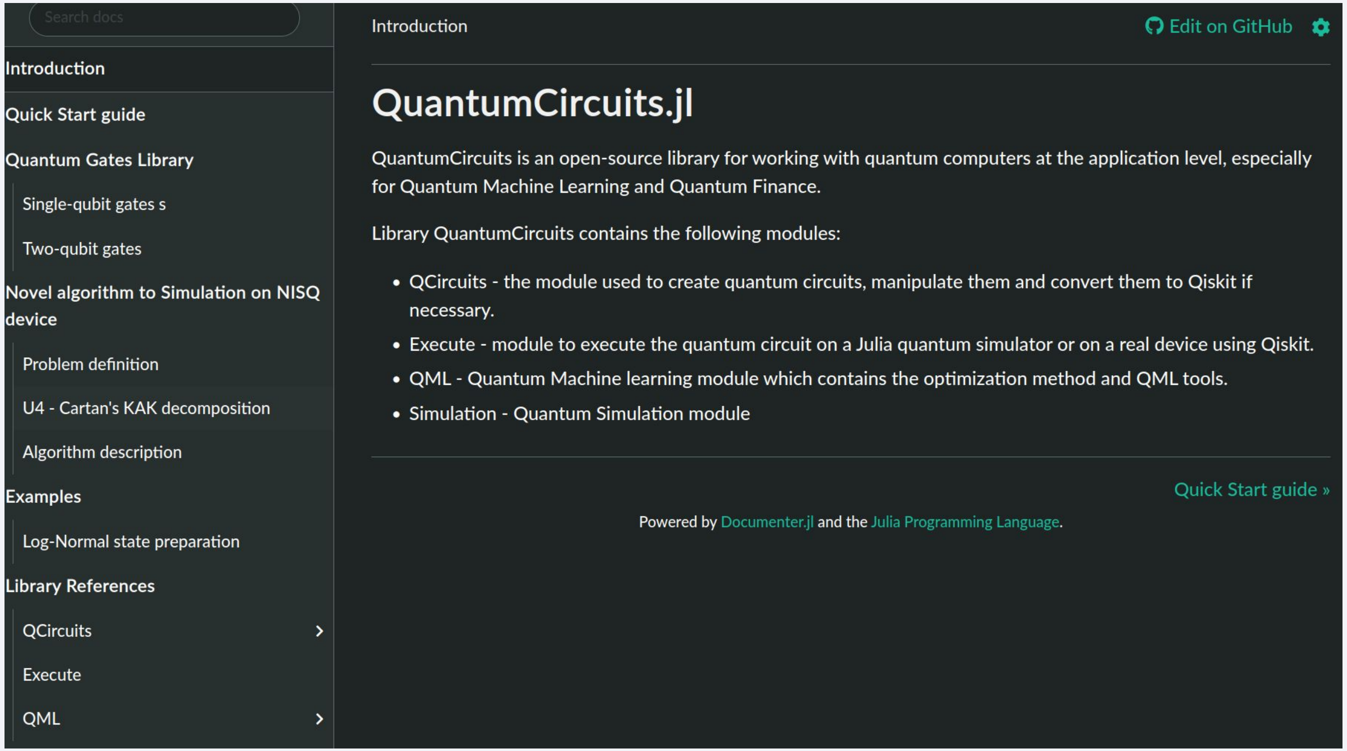




see: https://adgnitio.github.io/QuantumCircuits.jl/dev/Simulation/Trotterization/

Documentation

Qiskit



Introduction

Edit on GitHub

QuantumCircuits.jl

QuantumCircuits is an open-source library for working with quantum computers at the application level, especially for Quantum Machine Learning and Quantum Finance.

Library QuantumCircuits contains the following modules:

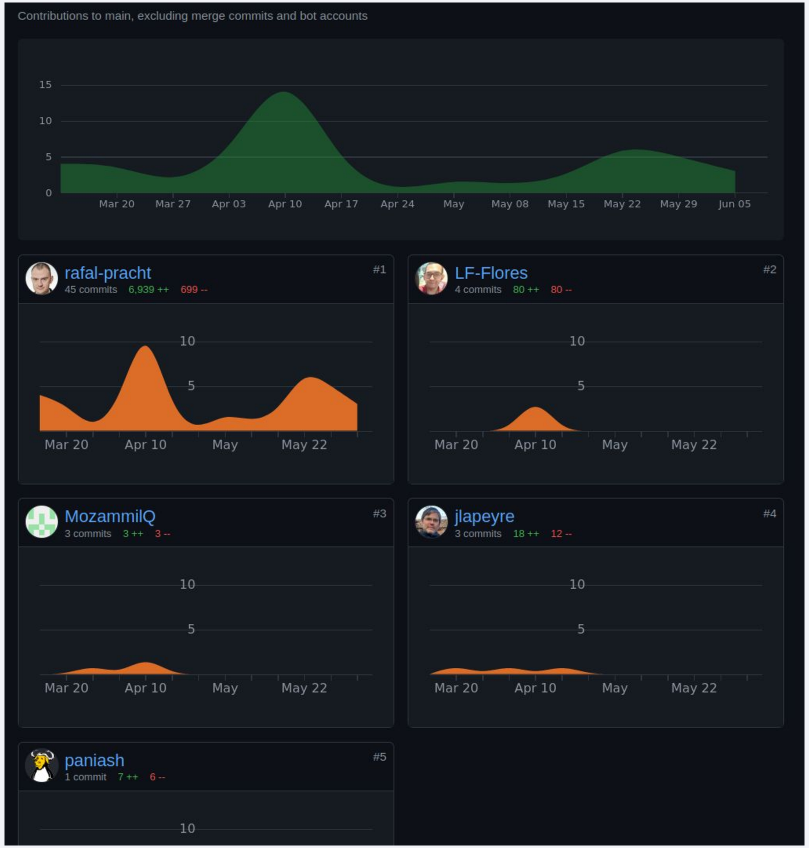
- QCircuits - the module used to create quantum circuits, manipulate them and convert them to Qiskit if necessary.
- Execute - module to execute the quantum circuit on a Julia quantum simulator or on a real device using Qiskit.
- QML - Quantum Machine learning module which contains the optimization method and QML tools.
- Simulation - Quantum Simulation module

Quick Start guide »

Powered by Documenter.jl and the Julia Programming Language.

see: https://adgnitio.github.io/QuantumCircuits.jl/dev/

Collaboration



Contributions to main, excluding merge commits and bot accounts

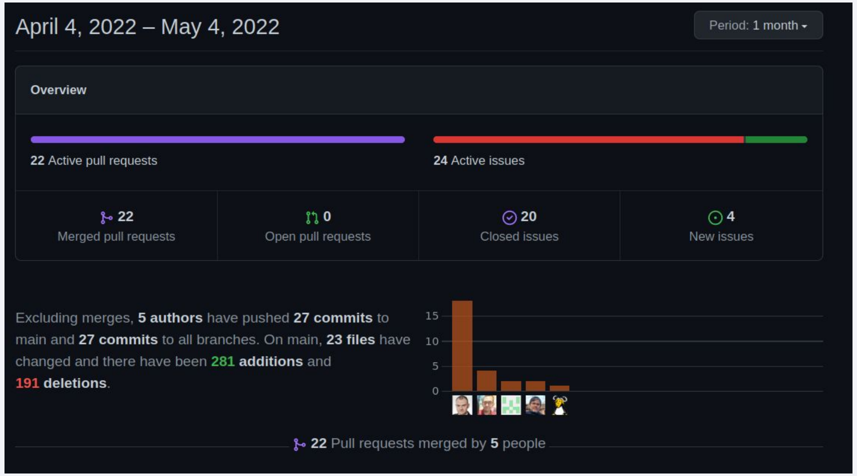
rafal-pracht #1
45 commits 6,939 ++ 699 --

LF-Flores #2
4 commits 80 ++ 80 --

MozammilQ #3
3 commits 3 ++ 3 --

jlapeyre #4
3 commits 18 ++ 12 --

paniash #5
1 commit 7 ++ 6 --



April 4, 2022 – May 4, 2022

Period: 1 month

Overview

22 Active pull requests

24 Active issues

22 Merged pull requests

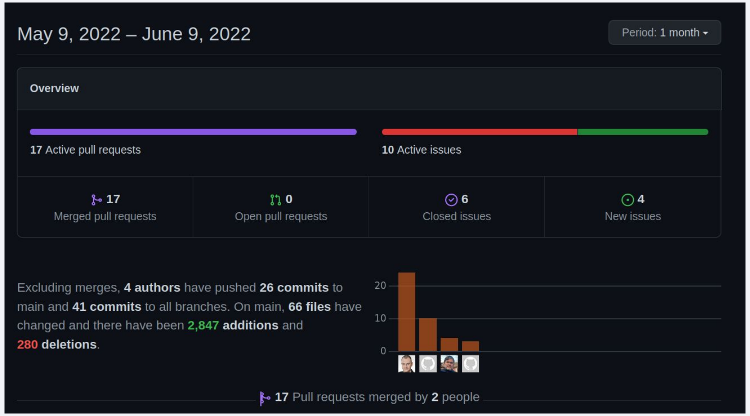
0 Open pull requests

20 Closed issues

4 New issues

Excluding merges, **5 authors** have pushed **27 commits** to main and **27 commits** to all branches. On main, **23 files** have changed and there have been **281 additions** and **191 deletions**.

22 Pull requests merged by **5** people



May 9, 2022 – June 9, 2022

Period: 1 month

Overview

17 Active pull requests

10 Active issues

17 Merged pull requests

0 Open pull requests

6 Closed issues

4 New issues

Excluding merges, **4 authors** have pushed **26 commits** to main and **41 commits** to all branches. On main, **66 files** have changed and there have been **2,847 additions** and **280 deletions**.

17 Pull requests merged by **2** people

see: https://adgnitio.github.io/QuantumCircuits.jl/dev/

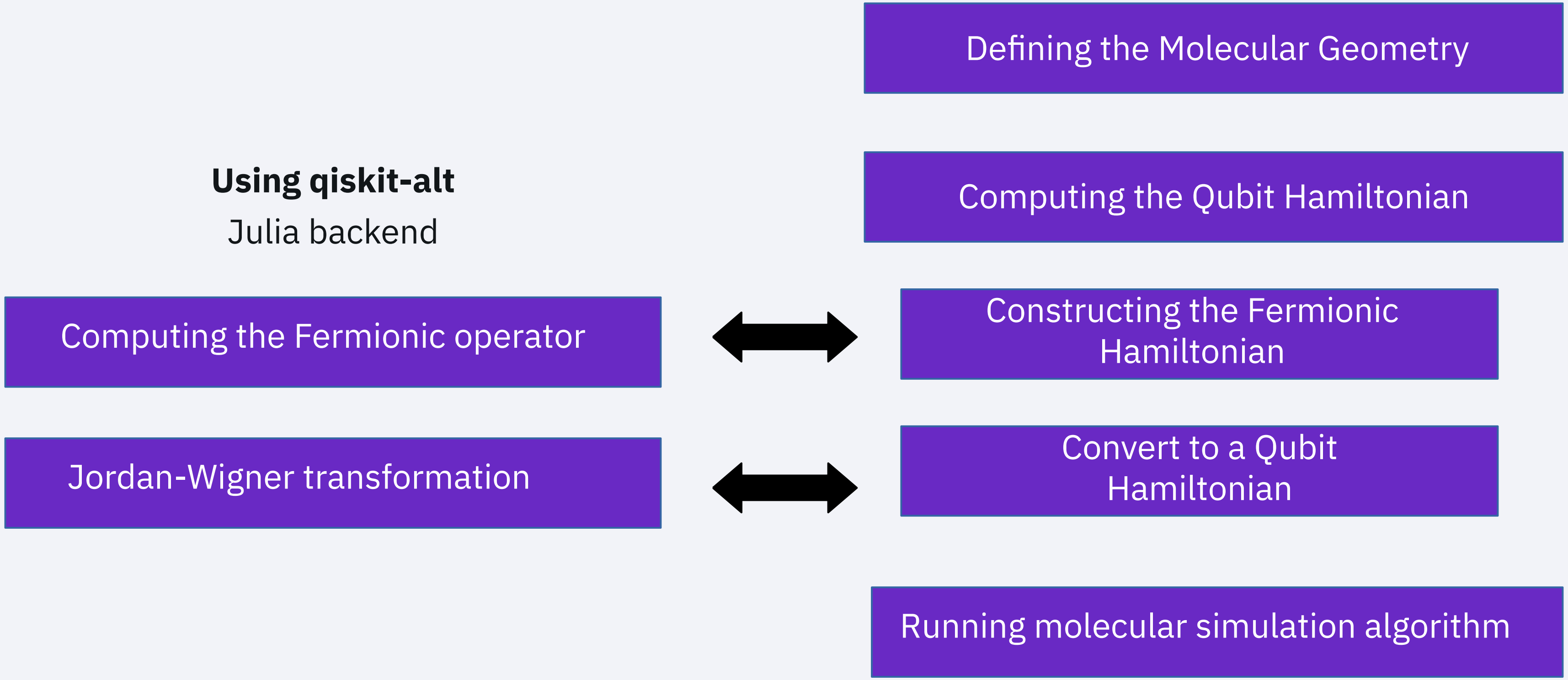
Qiskit-Alt

Qiskit-Alt workflow

Qiskit

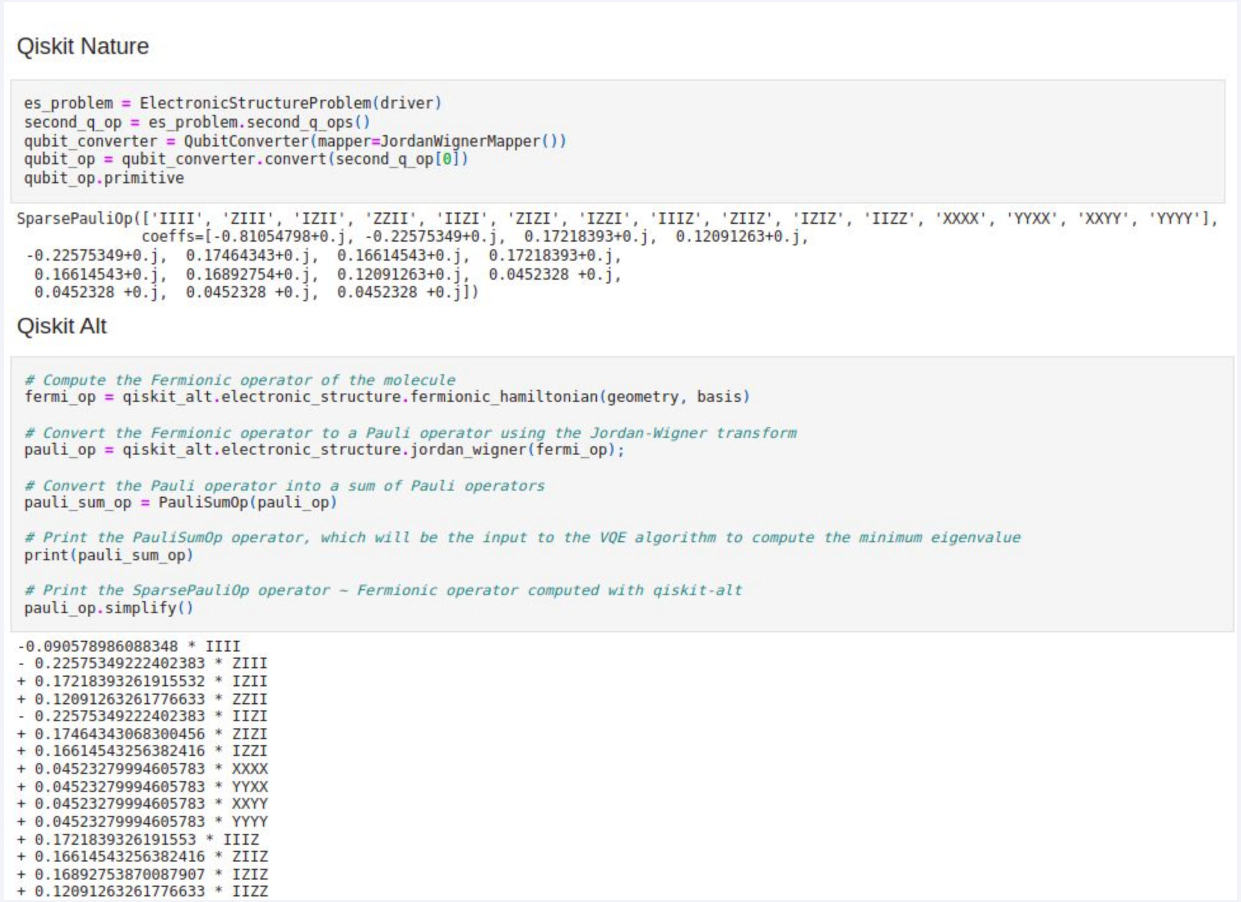
Using qiskit-alt

Julia backend


Defining the Molecular Geometry
Computing the Qubit Hamiltonian
Computing the Fermionic operator
Constructing the Fermionic Hamiltonian
Jordan-Wigner transformation
Convert to a Qubit Hamiltonian
Running molecular simulation algorithm

Computing the Qubit Hamiltonian: H_2 Molecule

Qiskit



Qiskit Nature

```
es_problem = ElectronicStructureProblem(driver)
second_q_op = es_problem.second_q_ops()
qubit_converter = QubitConverter(mapper=JordanWignerMapper())
qubit_op = qubit_converter.convert(second_q_op[0])
qubit_op.primitive
```

```
SparsePauliOp(['IIII', 'ZIII', 'IZII', 'ZZII', 'IIZI', 'ZIZI', 'IZZI', 'IIIZ', 'ZIIZ', 'IZIZ', 'IIZZ', 'XXXX', 'YYXX', 'XXYY', 'YYYY'],
              coeffs=[-0.81054798+0.j, -0.22575349+0.j,  0.17218393+0.j,  0.12091263+0.j,
 -0.22575349+0.j,  0.17464343+0.j,  0.16614543+0.j,  0.17218393+0.j,
  0.16614543+0.j,  0.16892754+0.j,  0.12091263+0.j,  0.0452328 +0.j,
  0.0452328 +0.j,  0.0452328 +0.j,  0.0452328 +0.j])
```

Qiskit Alt

```
# Compute the Fermionic operator of the molecule
fermi_op = qiskit_alt.electronic_structure.fermionic_hamiltonian(geometry, basis)

# Convert the Fermionic operator to a Pauli operator using the Jordan-Wigner transform
pauli_op = qiskit_alt.electronic_structure.jordan_wigner(fermi_op);

# Convert the Pauli operator into a sum of Pauli operators
pauli_sum_op = PauliSumOp(pauli_op)

# Print the PauliSumOp operator, which will be the input to the VQE algorithm to compute the minimum eigenvalue
print(pauli_sum_op)

# Print the SparsePauliOp operator ~ Fermionic operator computed with qiskit-alt
pauli_op.simplify()
```

```
-0.090578986088348 * IIII
- 0.22575349222402383 * ZIII
+ 0.17218393261915532 * IZII
+ 0.12091263261776633 * ZZII
- 0.22575349222402383 * IIZI
+ 0.17464343068300456 * ZIZI
+ 0.16614543256382416 * IZZI
+ 0.04523279994605783 * XXXX
+ 0.04523279994605783 * YYXX
+ 0.04523279994605783 * XXYY
+ 0.04523279994605783 * YYYY
+ 0.1721839326191553 * IIIZ
+ 0.16614543256382416 * ZIIZ
+ 0.16892753870087907 * IZIZ
+ 0.12091263261776633 * IIZZ
```

➡ **Similar generated Hamiltonian**

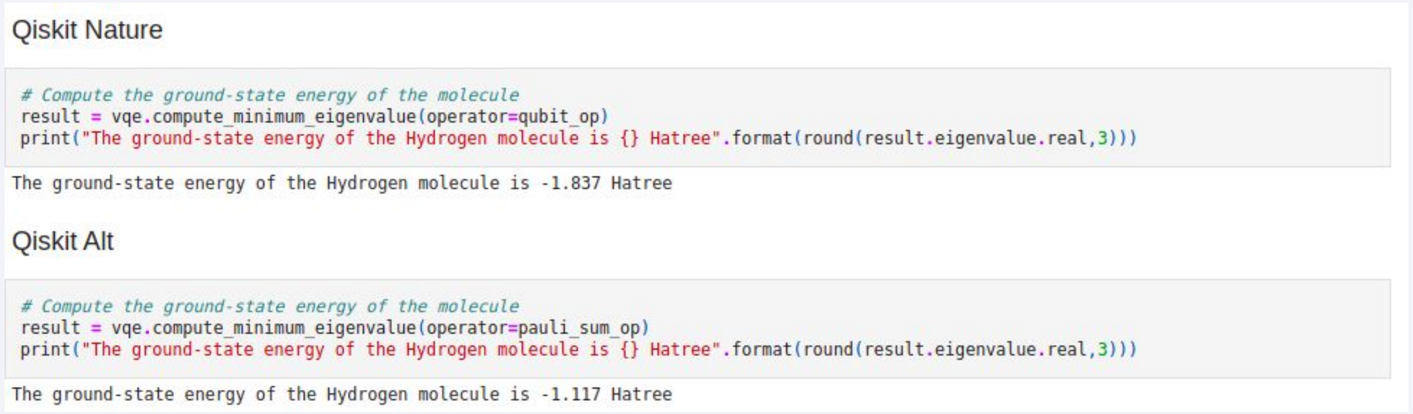
Running molecular simulation algorithm

Qiskit

Computing the ground state energy

- Ansatz: TwoLocal
- Optimizer: COBYLA
- Backend: Statevector Simulator

Results



Qiskit Nature

```
# Compute the ground-state energy of the molecule
result = vqe.compute_minimum_eigenvalue(operator=qubit_op)
print("The ground-state energy of the Hydrogen molecule is {} Hatree".format(round(result.eigenvalue.real,3)))
```

```
The ground-state energy of the Hydrogen molecule is -1.837 Hatree
```

Qiskit Alt

```
# Compute the ground-state energy of the molecule
result = vqe.compute_minimum_eigenvalue(operator=pauli_sum_op)
print("The ground-state energy of the Hydrogen molecule is {} Hatree".format(round(result.eigenvalue.real,3)))
```

```
The ground-state energy of the Hydrogen molecule is -1.117 Hatree
```

✓ Time advantage

✗ Results accuracy

Project plan

Qiskit

Objective: Integrate qiskit-alt in qiskit nature workflow

- ✓ Generate Hamiltonian operators with qiskit-alt
- ✓ VQE tutorial

- ✓ Features of Qiskit Nature
- ✗ Testing molecular simulation algorithms
- ✓ Testing performance

- ✓ Release a plugin to use qiksit-alt in qiskit nature workflow