# Package 'qwalkr'

May 9, 2026

**Title** Handle Continuous-Time Quantum Walks with R

**Version** 0.1.0

**Description** Functions and tools for creating, visualizing, and investigating properties of continuous-time quantum walks, including efficient calculation of matrices such as the mixing matrix, average mixing matrix, and spectral decomposition of the Hamiltonian. E. Farhi (1997): <doi:10.48550/arXiv.quant-ph/9706062v2>; C. Godsil (2011) <doi:10.48550/arXiv.1103.2578v3>.

**License** MIT + file LICENSE

**URL** https://github.com/vitormarquesr/qwalkr, https://vitormarquesr.github.io/qwalkr/

**BugReports** https://github.com/vitormarquesr/qwalkr/issues

**Encoding** UTF-8

**RoxygenNote** 7.2.3

**Suggests** knitr, rmarkdown, testthat (>= 3.0.0)

**Config/testthat/edition** 3

**VignetteBuilder** knitr

**Imports** lifecycle

**NeedsCompilation** no

**Author** Vitor Marques [aut, cre, cph]

**Maintainer** Vitor Marques <vmrodriguespro@gmail.com>

**Repository** CRAN

**Date/Publication** 2023-09-27 08:50:02 UTC

## Contents

---

`act_eigfun` *Apply a Function to an Operator*

---

## Description

Apply a Function to an Operator

## Usage

```
act_eigfun(object, ...)
```

## Arguments

| | |
|---|---|
| `object` | a representation of the operator. |
| `...` | further arguments passed to or from other methods. |

## Value

The resulting operator from the application of the function.

## See Also

`act_eigfun.spectral()`

## Examples

```
s <- spectral(rbind(c(0.5, 0.3), c(0.3,0.7)))

act_eigfun(s, function(x) x^2) #-> act_eigfun.spectral(...)
```

---

`act_eigfun.spectral` *Apply a Function to a Hermitian Matrix*

---

## Description

Apply a function to a Hermitian matrix based on the representation given by class `spectral`.

## Usage

```
## S3 method for class 'spectral'
act_eigfun(object, FUN, ...)
```

## Arguments

| | |
|---|---|
| `object` | an instance of class `spectral`. |
| `FUN` | the function to be applied to the matrix. |
| `...` | further arguments passed on to `FUN`. |

## Value

The matrix resulting from the application of `FUN`.

A Hermitian Matrix admits the spectral decomposition

$$H = \sum_k \lambda_k E_k$$

where $\lambda_k$ are its eigenvalues and $E_k$ the orthogonal projector onto the $\lambda_k$-eigenspace.

If $f$=`FUN` is defined on the eigenvalues of `H`, then `act_eigfun` performs the following calculation

$$f(H) = \sum_k f(\lambda_k) E_k$$

## See Also

`spectral()`, `act_eigfun()`

## Examples

```
H <- matrix(c(0,1,1,1,0,1,1,1,0), nrow=3)
decomp <- spectral(H)

# Calculates H^2.
act_eigfun(decomp, FUN = function(x) x^2)

# Calculates sin(H).
act_eigfun(decomp, FUN = function(x) sin(x))

# Calculates H^3.
act_eigfun(decomp, FUN = function(x, y) x^y, 3)
```

---

`avg_matrix` *The Average Mixing Matrix of a Quantum Walk*

---

## Description

The Average Mixing Matrix of a Quantum Walk

## Usage

```
avg_matrix(object, ...)
```

## Arguments

| | |
|---|---|
| `object` | a representation of the quantum walk. |
| `...` | further arguments passed to or from other methods. |

## Value

The average mixing matrix.

## See Also

`mixing_matrix()`, `gavg_matrix()`, `avg_matrix.ctqwalk()`

## Examples

```
w <- ctqwalk(matrix(c(0,1,0,1,0,1,0,1,0), nrow=3))

avg_matrix(w) #-> avg_matrix.ctqwalk(...)
```

avg_matrix.ctqwalk *The Average Mixing Matrix of a Continuous-Time Quantum Walk*

## Description

The Average Mixing Matrix of a Continuous-Time Quantum Walk

## Usage

```
## S3 method for class 'ctqwalk'
avg_matrix(object, ...)
```

## Arguments

| | |
|---|---|
| object | a representation of the quantum walk. |
| ... | further arguments passed to or from other methods. |

## Details

Let $M(t)$ be the mixing matrix of the quantum walk, then the average mixing matrix is defined as

$$\widehat{M} := \lim_{T\to\infty} \frac{1}{T} \int_0^T M(t)\mathrm{d}t$$

and encodes the long-term average behavior of the walk. Given the Hamiltonian $H = \sum_r \lambda_r E_r$, it is possible to prove that

$$\widehat{M} = \sum_r E_r \circ E_r$$

## Value

`avg_matrix()` returns the average mixing matrix as a square matrix of the same order as the walk.

## See Also

`ctqwalk()`, `avg_matrix()`

## Examples

```
walk <- ctqwalk(matrix(c(0,1,0,1,0,1,0,1,0), nrow=3))

# Return the average mixing matrix
avg_matrix(walk)
```

`cartesian` *Adjacency Matrix of the Cartesian Product*

## Description

Returns the adjacency matrix of the cartesian product of two graphs given the adjacency matrix of each one, $G$ and $H$.

## Usage

```
cartesian(G, H = NULL)
```

## Arguments

| | |
|---|---|
| `G` | adjacency matrix of the first graph. |
| `H` | adjacency matrix of the second graph. If not provided, it takes the same value as `G`. |

## Value

Let $A(G)$, $A(H)$ be the adjacency matrices of the graphs $G$, $H$ such that $|V(G)| = n$ and $|V(H)| = m$, then the adjacency matrix of the cartesian product $G \times H$ is given by

$$A(G \times H) = A(G) \otimes I_{m\ x\ m} + I_{n\ x\ n} \otimes A(H)$$

## See Also

`J()`, `tr()`, `trdot()`

## Examples

```
P3 <- matrix(c(0,1,0,1,0,1,0,1,0), nrow=3)
K3 <- matrix(c(0,1,1,1,0,1,1,1,0), nrow=3)

# Return the adjacency matrix of P3 X K3
cartesian(P3, K3)

# Return the adjacency matrix of P3 X P3
cartesian(P3)
```

## ctqwalk *Create a Continuous-time Quantum Walk*

### Description

`ctqwalk()` creates a quantum walk object from a hamiltonian.

### Usage

```
ctqwalk(hamiltonian, ...)
```

### Arguments

| | |
|---|---|
| `hamiltonian` | a Hermitian Matrix representing the Hamiltonian of the system. |
| `...` | further arguments passed on to `spectral()` |

### Value

A list with the walk related objects, i.e the hamiltonian and its spectral decomposition (See `spectral()` for further details)

### See Also

`spectral()`, `unitary_matrix.ctqwalk()`, `mixing_matrix.ctqwalk()`, `avg_matrix.ctqwalk()`, `gavg_matrix.ctqwalk()`

### Examples

```
# Creates a walk from the adjacency matrix of the graph P3.
ctqwalk(matrix(c(0,1,0,1,0,1,0,1,0), nrow=3))
```

## gavg_matrix *The Generalized Average Mixing Matrix of a Quantum Walk*

### Description

The Generalized Average Mixing Matrix of a Quantum Walk

### Usage

```
gavg_matrix(object, ...)
```

### Arguments

| | |
|---|---|
| `object` | a representation of the quantum walk. |
| `...` | further arguments passed to or from other methods. |

### Value

The generalized average mixing matrix.

### See Also

`mixing_matrix()`, `avg_matrix()`, `gavg_matrix.ctqwalk()`

### Examples

```
w <- ctqwalk(matrix(c(0,1,0,1,0,1,0,1,0), nrow=3))

gavg_matrix(w, rnorm(100)) #-> gavg_matrix.ctqwalk(...)
```

---

## `gavg_matrix.ctqwalk` *The Generalized Average Mixing Matrix of a Continuous-Time Quantum Walk*

---

### Description

The Generalized Average Mixing Matrix of a Continuous-Time Quantum Walk

### Usage

```
## S3 method for class 'ctqwalk'
gavg_matrix(object, R, ...)
```

### Arguments

| | |
|---|---|
| `object` | a representation of the quantum walk. |
| `R` | samples from the random variable $R$ (For performance, it is recommended at most 10000 samples). |
| `...` | further arguments passed to or from other methods. |

### Details

Let $M(t)$ be the mixing matrix of the quantum walk and $R$ a random variable with associated probability density function $f_R(t)$. Then the generalized average mixing matrix under $R$ is defined as

$$\widehat{M}_R := \mathbb{E}[M(R)] = \int_{-\infty}^{\infty} M(t) f_R(t) \mathrm{d}t$$

## Value

gavg_matrix() returns the generalized average mixing matrix as a square matrix of the same order as the walk.

## See Also

ctqwalk(), gavg_matrix()

## Examples

```
walk <- ctqwalk(matrix(c(0,1,0,1,0,1,0,1,0), nrow=3))

# Return the average mixing matrix under a Standard Gaussian distribution
gavg_matrix(walk, rnorm(1000))
```

---

# get_eigproj *Extract an Eigen-Projector from an operator*

---

## Description

Extract an Eigen-Projector from an operator

## Usage

```
get_eigproj(object, ...)
```

## Arguments

| | |
|---|---|
| object | a representation of the operator. |
| ... | further arguments passed to or from other methods. |

## Value

A representation of the requested eigen-projector.

## See Also

get_eigspace(), get_eigschur(), get_eigproj.spectral()

## Examples

```
s <- spectral(rbind(c(0.5, 0.3), c(0.3,0.7)))

get_eigproj(s, 1) #-> get_eigproj.spectral(...)
```

`get_eigproj.spectral` *Extract an Eigen-Projector from a Hermitian Matrix*

## Description

Get the orthogonal projector associated with an eigenspace based on the representation of a Hermitian Matrix given by class `spectral`.

## Usage

```
## S3 method for class 'spectral'
get_eigproj(object, id, ...)
```

## Arguments

| | |
|---|---|
| `object` | an instance of class `spectral`. |
| `id` | index for the desired eigenspace according to the ordered (decreasing) spectra. |
| `...` | further arguments passed to or from other methods. |

## Value

The orthogonal projector of the desired eigenspace.

A Hermitian matrix `S` admits the spectral decomposition $S = \sum_r \lambda_r E_r$ such that $E_r$ is the orthogonal projector onto the $\lambda_r$-eigenspace. If $V_{id}$ is the matrix associated to the eigenspace, then

$$E_{id} = V_{id} V_{id}^*$$

## See Also

`spectral()`, `get_eigproj()`

## Examples

```
# Spectra is {2, -1} with multiplicities one and two respectively.
decomp <- spectral(matrix(c(0,1,1,1,0,1,1,1,0), nrow=3))

# Returns the projector associated to the eigenvalue -1.
get_eigproj(decomp, id=2)

# Returns the projector associated to the eigenvalue 2.
get_eigproj(decomp, id=1)
```

## get_eigschur *Extract a Schur Cross-Product from an Operator*

### Description

Extract a Schur Cross-Product from an Operator

### Usage

```
get_eigschur(object, ...)
```

### Arguments

| | |
|---|---|
| object | a representation of the operator. |
| ... | further arguments passed to or from other methods. |

### Value

A representation of the requested Schur cross-product.

### See Also

get_eigspace(), get_eigproj(), get_eigschur.spectral()

### Examples

```
s <- spectral(rbind(c(0.5, 0.3), c(0.3,0.7)))

get_eigschur(s, 1, 2) #-> get_eigschur.spectral(...)
```

## get_eigschur.spectral *Extract a Schur Cross-Product from a Hermitian Matrix*

### Description

Get the Schur product between eigen-projectors based on the representation of a Hermitian Matrix given by class spectral.

### Usage

```
## S3 method for class 'spectral'
get_eigschur(object, id1, id2 = NULL, ...)
```

### Arguments

| | |
|---|---|
| object | an instance of class spectral. |
| id1 | index for the first eigenspace according to the ordered (decreasing) spectra. |
| id2 | index for the second eigenspace according to the ordered (decreasing) spectra. If not provided, it takes the same value as id1. |
| ... | further arguments passed to or from other methods. |

### Value

The Schur product of the corresponding eigenprojectors, $E_{id_1} \circ E_{id_2}$.

### See Also

spectral(), get_eigschur()

### Examples

```
# Spectra is {2, -1} with multiplicities one and two respectively.
decomp <- spectral(matrix(c(0,1,1,1,0,1,1,1,0), nrow=3))

# Returns the Schur product between the 2-projector and -1-projector.
get_eigschur(decomp, id1=2, id2=1)

# Returns the Schur square of the 2-projector.
get_eigschur(decomp, id1=1, id2=1)

# Also returns the Schur square of the 2-projector
get_eigschur(decomp, id1=1)
```

---

## get_eigspace *Extract an Eigenspace from an Operator*

---

### Description

Extract an Eigenspace from an Operator

### Usage

```
get_eigspace(object, ...)
```

### Arguments

| | |
|---|---|
| object | a representation of the operator. |
| ... | further arguments passed to or from other methods. |

## Value

A representation of the requested eigenspace.

## See Also

get_eigproj(), get_eigschur(), get_eigspace.spectral()

## Examples

```
s <- spectral(rbind(c(0.5, 0.3), c(0.3,0.7)))

get_eigspace(s, 1) #-> get_eigspace.spectral(...)
```

---

# get_eigspace.spectral *Extract an Eigenspace from a Hermitian Matrix*

---

## Description

Get the eigenbasis associated with an eigenvalue based on the representation of a Hermitian Matrix given by class `spectral`.

## Usage

```
## S3 method for class 'spectral'
get_eigspace(object, id, ...)
```

## Arguments

| | |
|---|---|
| `object` | an instance of class `spectral`. |
| `id` | index for the desired eigenspace according to the ordered (decreasing) spectra. |
| `...` | further arguments passed to or from other methods. |

## Value

A matrix whose columns form the orthonormal eigenbasis.

If `s <- spectral(A)` and `V <- s$eigvectors`, then the extracted eigenspace $V_{id}$ is some submatrix `V[, _]`.

## See Also

spectral(), get_eigspace()

## Examples

```
# Spectra is {2, -1} with multiplicities one and two respectively.
decomp <- spectral(matrix(c(0,1,1,1,0,1,1,1,0), nrow=3))

# Returns the two orthonormal eigenvectors corresponding to the eigenvalue -1.
get_eigspace(decomp, id=2)

# Returns the eigenvector corresponding to the eigenvalue 2.
get_eigspace(decomp, id=1)
```

---

J *The All-Ones Matrix*

---

## Description

Returns the all-ones matrix of order n.

## Usage

```
J(n)
```

## Arguments

| | |
|---|---|
| n | the order of the matrix. |

## Value

A square matrix of order $n$ in which every entry is equal to 1. The all-ones matrix is given by $J_{n\ x\ n} = 1_{n\ x\ 1} 1_{n\ x\ 1}^T$.

## See Also

tr(), trdot(), cartesian()

## Examples

```
# Return the all-ones matrix of order 5.
J(5)
```

## mixing_matrix *The Mixing Matrix of a Quantum Walk*

### Description

The Mixing Matrix of a Quantum Walk

### Usage

```
mixing_matrix(object, ...)
```

### Arguments

| | |
|---|---|
| object | a representation of the quantum walk. |
| ... | further arguments passed to or from other methods. |

### Value

The mixing matrix of the quantum walk.

### See Also

unitary_matrix(), avg_matrix(), gavg_matrix(), mixing_matrix.ctqwalk()

### Examples

```
w <- ctqwalk(matrix(c(0,1,0,1,0,1,0,1,0), nrow=3))

mixing_matrix(w, t = 2*pi) #-> mixing_matrix.ctqwalk(...)
```

## mixing_matrix.ctqwalk *The Mixing Matrix of a Continuous-Time Quantum Walk*

### Description

The Mixing Matrix of a Continuous-Time Quantum Walk

### Usage

```
## S3 method for class 'ctqwalk'
mixing_matrix(object, t, ...)
```

### Arguments

| | |
|---|---|
| `object` | an instance of class `ctqwalk`. |
| `t` | it will be returned the mixing matrix at time `t`. |
| `...` | further arguments passed to or from other methods. |

### Details

Let $U(t)$ be the time evolution operator of the quantum walk at time $t$, then the mixing matrix is given by

$$M(t) = U(t) \circ \overline{U(t)}$$

$M(t)$ is a doubly stochastic real symmetric matrix, which encodes the probability density of the quantum system at time $t$.

More precisely, the $(M(t))_{ab}$ entry gives us the probability of measuring the standard basis state $|b\rangle$ at time $t$, given that the quantum walk started at $|a\rangle$.

### Value

`mixing_matrix()` returns the mixing matrix of the CTQW evaluated at time `t`.

### See Also

`ctqwalk()`, `mixing_matrix()`

### Examples

```
walk <- ctqwalk(matrix(c(0,1,0,1,0,1,0,1,0), nrow=3))

# Returns the mixing matrix at time t = 2*pi, M(2pi)
mixing_matrix(walk, t = 2*pi)
```

---

## print.ctqwalk *Print the ctqwalk output*

---

### Description

Print the ctqwalk output

### Usage

```
## S3 method for class 'ctqwalk'
print(x, ...)
```

### Arguments

| | |
|---|---|
| x | an object of the class ctqwalk. |
| ... | further arguments passed to or from other methods. |

### Value

Called mainly for its side effects. However, also returns x invisibly.

---

## spectral *Spectral Decomposition of a Hermitian Matrix*

---

### Description

spectral() is a wrapper around base::eigen() designed for Hermitian matrices, which can handle repeated eigenvalues.

### Usage

```
spectral(S, multiplicity = TRUE, tol = .Machine$double.eps^0.5, ...)
```

### Arguments

| | |
|---|---|
| S | a Hermitian matrix. *Obs*: The matrix is always assumed to be Hermitian, and only its lower triangle (diagonal included) is used. |
| multiplicity | if TRUE (default), tries to infer eigenvalue multiplicity. If set to FALSE, each eigenvalue is considered unique with multiplicity one. |
| tol | two eigenvalues x, y are considered equal if abs(x-y) < tol. Defaults to tol=.Machine$double.eps^0. |
| ... | further arguments passed on to base::eigen() |

### Value

The spectral decomposition of S is returned as a list with components

| | |
|---|---|
| eigvals | vector containing the unique eigenvalues of S in *decreasing* order. |
| multiplicity | multiplicities of the eigenvalues in eigvals. |
| eigvectors | a nrow(S) x nrow(S) unitary matrix whose columns are eigenvectors ordered according to eigvals. Note that there may be more eigenvectors than eigenvalues if multiplicity=TRUE, however eigenvectors of the same eigenspace are next to each other. |

The Spectral Theorem ensures the eigenvalues of S are real and that the vector space admits an orthonormal basis consisting of eigenvectors of S. Thus, if s <- spectral(S), and V <- s$eigvectors; lam <- s$eigvals, then

$$S = V \Lambda V^*$$

where $\Lambda =$ diag(rep(lam, times=s$multiplicity))

## See Also

base::eigen(), get_eigspace.spectral(), get_eigproj.spectral(), get_eigschur.spectral(), act_eigfun.spectral()

## Examples

```
spectral(matrix(c(0,1,0,1,0,1,0,1,0), nrow=3))

# Use "tol" to set the tolerance for numerical equality
spectral(matrix(c(0,1,0,1,0,1,0,1,0), nrow=3), tol=10e-5)

# Use "multiplicity=FALSE" to force each eigenvalue to be considered unique
spectral(matrix(c(0,1,0,1,0,1,0,1,0), nrow=3), multiplicity = FALSE)
```

---

# tr *The Trace of a Matrix*

---

## Description

Computes the trace of a matrix A.

## Usage

```
tr(A)
```

## Arguments

| | |
|---|---|
| A | a square matrix. |

## Value

If $A$ has order $n$, then $tr(A) = \sum_{i=1}^{n} a_{ii}$.

## See Also

J(), trdot(), cartesian()

## Examples

```
A <- rbind(1:5, 2:6, 3:7)

# Calculate the trace of A
tr(A)
```

## trdot *The Trace Inner Product of Matrices*

### Description

Computes the trace inner product of two matrices A and B.

### Usage

```
trdot(A, B)
```

### Arguments

| | |
|---|---|
| A, B | square matrices. |

### Value

The trace inner product on $Mat_{n\ x\ n}(\mathbb{C})$ is defined as

$$\langle A, B\rangle := tr(A^* B)$$

### See Also

J(), tr(), cartesian()

### Examples

```
A <- rbind(1:5, 2:6, 3:7)
B <- rbind(7:11, 8:12, 9:13)

# Compute the trace inner product of A and B
trdot(A, B)
```

## unitary_matrix *The Unitary Time Evolution Operator of a Quantum Walk*

### Description

The Unitary Time Evolution Operator of a Quantum Walk

### Usage

```
unitary_matrix(object, ...)
```

### Arguments

| | |
|---|---|
| `object` | a representation of the quantum walk. |
| `...` | further arguments passed to or from other methods. |

### Value

The unitary time evolution operator.

### See Also

`mixing_matrix()`, `unitary_matrix.ctqwalk()`

### Examples

```
w <- ctqwalk(matrix(c(0,1,0,1,0,1,0,1,0), nrow=3))

unitary_matrix(w, t = 2*pi) #-> unitary_matrix.ctqwalk(...)
```

---

## `unitary_matrix.ctqwalk` *The Unitary Time Evolution Operator of a Continuous-Time Quantum Walk*

---

### Description

The Unitary Time Evolution Operator of a Continuous-Time Quantum Walk

### Usage

```
## S3 method for class 'ctqwalk'
unitary_matrix(object, t, ...)
```

### Arguments

| | |
|---|---|
| `object` | an instance of class `ctqwalk`. |
| `t` | it will be returned the evolution operator at time `t`. |
| `...` | further arguments passed to or from other methods. |

### Details

If $|\psi(t)\rangle$ is the quantum state of the system at time $t$, and $H$ the Hamiltonian operator, then the evolution is governed by the Schrodinger equation

$$\frac{\partial}{\partial t}|\psi(t)\rangle = iH|\psi(t)\rangle$$

and if $H$ is time-independent its solution is given by

$$|\psi(t)\rangle = U(t)|\psi(0)\rangle = e^{iHt}|\psi(0)\rangle$$

The evolution operator is the result of the complex matrix exponential and it can be calculated as

$$U(t) = e^{iHt} = \sum_r e^{it\lambda_r} E_r$$

in which $H = \sum_r \lambda_r E_r$.

## Value

`unitary_matrix()` returns the unitary time evolution operator of the CTQW evaluated at time `t`.

## See Also

`ctqwalk()`, `unitary_matrix()`, `act_eigfun()`

## Examples

```
walk <- ctqwalk(matrix(c(0,1,0,1,0,1,0,1,0), nrow=3))

# Returns the operator at time t = 2*pi, U(2pi)
unitary_matrix(walk, t = 2*pi)
```

# Index