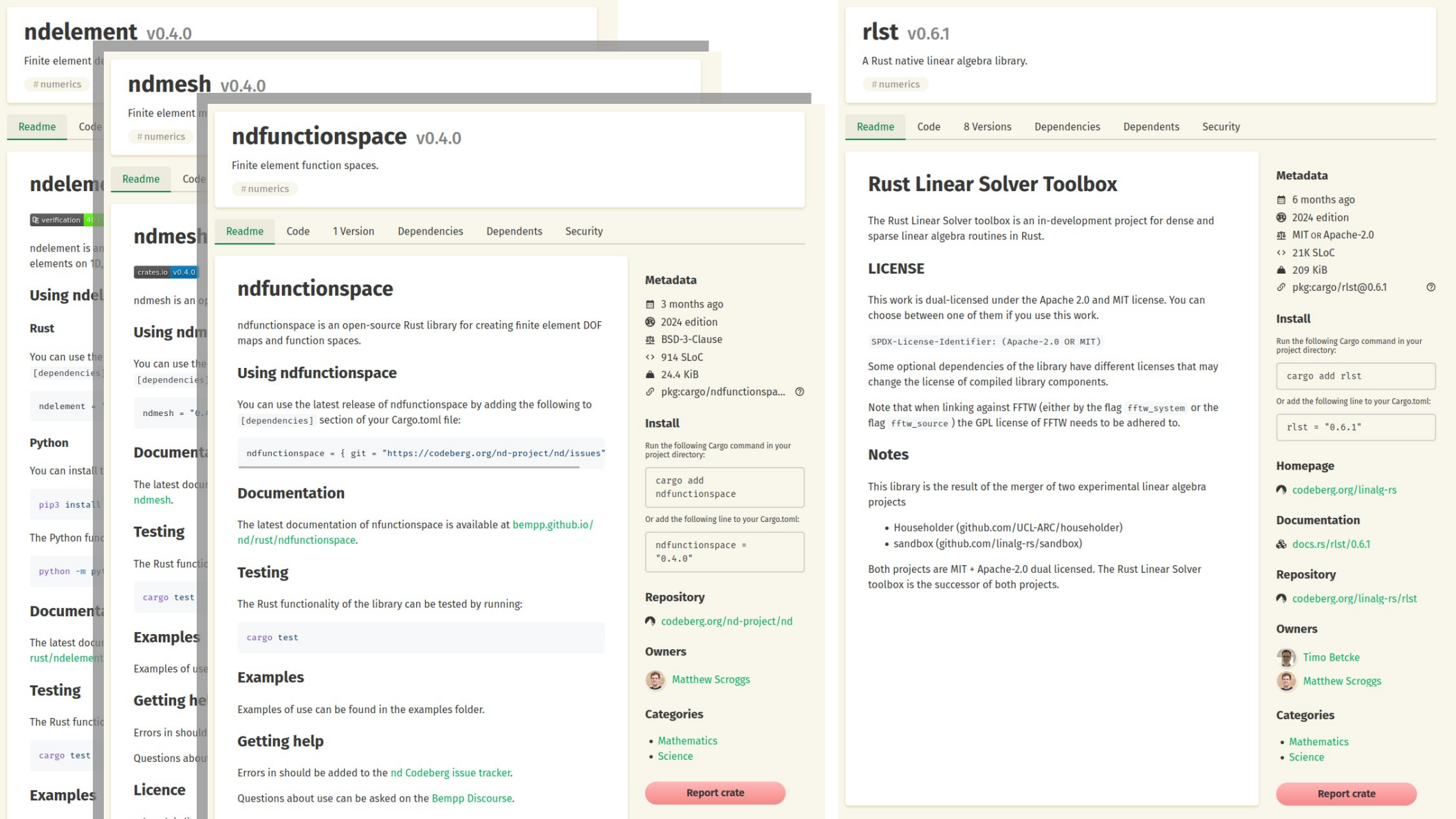
cargo-templated-examples

Matthew Scroggs
Advanced Research Computing,
University College London

mscroggs.co.uk
matthew.scroggs.14@ucl.ac.uk
mscroggs
@mscroggs@mathstodon.xyz
@mscroggs.co.uk



ndelement v0.4.0

Finite element

#numerics

ndmesh v0.4.0

Finite element

#numerics

Readme Code

ndfunctionspace v0.4.0

Finite element function spaces.

#numerics

Readme Code 1 Version Dependencies Dependents Security

ndfunctionspace

ndfunctionspace is an open-source Rust library for creating finite element DOF maps and function spaces.

Using ndfunctionspace

You can use the latest release of ndfunctionspace by adding the following to [dependencies] section of your Cargo.toml file:

```
ndfunctionspace = { git = "https://codeberg.org/nd-project/nd/issues"
```

Documentation

The latest documentation of nfunctionspace is available at bempp.github.io/nd/rust/ndfunctionspace.

Testing

The Rust functionality of the library can be tested by running:

```
cargo test
```

Examples

Examples of use can be found in the examples folder.

Getting help

Errors in should be added to the nd Codeberg issue tracker.

Questions about use can be asked on the Bempp Discourse.

Metadata

3 months ago
2024 edition
BSD-3-Clause
914 SLoC
24.4 KiB
pkg:cargo/ndfunctionspa...

Install

Run the following Cargo command in your project directory:

```
cargo add ndfunctionspace
```

Or add the following line to your Cargo.toml:

```
ndfunctionspace = "0.4.0"
```

Repository

codeberg.org/nd-project/nd

Owners

Matthew Scroggs

Categories

- Mathematics
- Science

Report crate

rlst v0.6.1

A Rust native linear algebra library.

#numerics

Readme Code 8 Versions Dependencies Dependents Security

Rust Linear Solver Toolbox

The Rust Linear Solver toolbox is an in-development project for dense and sparse linear algebra routines in Rust.

LICENSE

This work is dual-licensed under the Apache 2.0 and MIT license. You can choose between one of them if you use this work.

```
SPDX-License-Identifier: (Apache-2.0 OR MIT)
```

Some optional dependencies of the library have different licenses that may change the license of compiled library components.

Note that when linking against FFTW (either by the flag `fftw_system` or the flag `fftw_source`) the GPL license of FFTW needs to be adhered to.

Notes

This library is the result of the merger of two experimental linear algebra projects

- Householder (github.com/UCL-ARC/householder)
- sandbox (github.com/linalg-rs/sandbox)

Both projects are MIT + Apache-2.0 dual licensed. The Rust Linear Solver toolbox is the successor of both projects.

Metadata

6 months ago
2024 edition
MIT OR Apache-2.0
21K SLoC
209 KiB
pkg:cargo/rlst@0.6.1

Install

Run the following Cargo command in your project directory:

```
cargo add rlst
```

Or add the following line to your Cargo.toml:

```
rlst = "0.6.1"
```

Homepage

codeberg.org/linalg-rs

Documentation

docs.rs/rlst/0.6.1

Repository

codeberg.org/linalg-rs/rlst

Owners

Timo Betcke
Matthew Scroggs

Categories

- Mathematics
- Science

Report crate

Examples

```
cargo run --example mixed_mesh --release
cargo run --example single_element_mesh --release
cargo run --example io --features "serde" --release
cargo mpirun -n 2 --example parallel_mesh --features "serde,scotch,mpi" --release
cargo mpirun -n 4 --example parallel_mesh --features "serde,scotch,mpi" --release
cargo mpirun -n 4 --example parallel_io --features "serde,mpi" --release
```

Goal

Create a new command that will run all of the above:

```
cargo templated-examples
```

How to make a cargo extension

- Create a crate called “cargo-your-command”:

```
cargo new --bin cargo-your-command
```

- The function `main` in main.rs will be run when a user runs `cargo your-command`
  - Users need to `cargo install cargo-your-command` before they can do this

cargo-templated-examples

```
fn main() → ExitCode {
    let dir = cargo_toml::find();
    let outcomes = run_all_examples(&dir, None);

    println!();
    println!("SUMMARY");
    if outcomes.passes + outcomes.fails == 0 {
        println!("Couldn't find any examples to run.");
        ExitCode::FAILURE
    } else {
        println!("{} example(s) ran successfully.", outcomes.passes);
        if outcomes.fails == 0 {
            ExitCode::SUCCESS
        } else {
            println!("{} example(s) encountered errors.", outcomes.fails);
            ExitCode::FAILURE
        }
    }
}
```

Search parent directories for Cargo.toml

Run all examples (this function's definition is quite long)

Return an ExitCode. This is necessary so that CI runs pick up failures

Using cargo-templated-examples

- Either:
  - Add a `//?` comment to the top of your file, eg:

```
//? mpirun -n 4
```

  - Add `templated-examples` metadata to Cargo.toml, eg:

```
[package.metadata.example.FILENAME.templated-examples]
command = "mpirun -n 4"
```

- `cargo templated-examples` will run:

```
cargo mpirun -n 4 --example FILENAME
```

Using cargo-templated-examples

- You can use variables, eg:

```
//? mpirun -n {{NPROCESSES}}
```

- Then `cargo templated-examples NPROCESSES 2,4` will run:

```
cargo mpirun -n 2 --example FILENAME --release
cargo mpirun -n 4 --example FILENAME --release
```

Using cargo-templated-examples

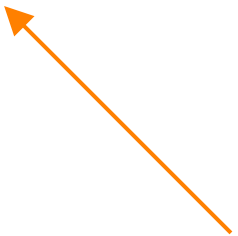
- `required-features` in Cargo.toml can be used to set the features for an example, eg:

```
[[example]]
name = "FILENAME"
required-features = ["serde", "mpi"]
```

- Then `cargo templated-examples` will run:

```
cargo run --example FILENAME --features "serde,mpi" --release
```

```
cargo run --example mixed_mesh --release
cargo run --example single_element_mesh --release
cargo run --example io --features "serde" --release
cargo mpirun -n 2 --example parallel_mesh --features "serde,scotch,mpi" --release
cargo mpirun -n 4 --example parallel_mesh --features "serde,scotch,mpi" --release
cargo mpirun -n 4 --example parallel_io --features "serde,mpi" --release
```



```
cargo templated-examples NPROCESSES 2,4
```

```
[[example]]
name = "io"
required-features = ["serde"]

[[example]]
name = "parallel_mesh"
required-features = ["serde", "scotch", "mpi"]

[package.metadata.example.parallel_mesh.templated-examples]
command = "mpirun -n {{NPROCESSES}}"

[[example]]
name = "parallel_io"
required-features = ["serde", "mpi"]

[package.metadata.example.parallel_io.templated-examples]
command = "mpirun -n 4"
```

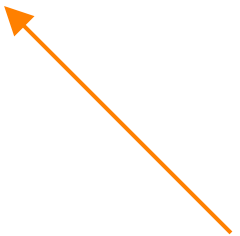
```
cargo run --example mixed_mesh --release
cargo run --example single_element_mesh --release
cargo run --example io --features "serde" --release
cargo mpirun -n 2 --example parallel_mesh --features "serde,scotch,mpi" --release
cargo mpirun -n 4 --example parallel_mesh --features "serde,scotch,mpi" --release
cargo mpirun -n 4 --example parallel_io --features "serde,mpi" --release
```

```
[[example]]
name = "io"
required-features = ["serde"]

[[example]]
name = "parallel_mesh"
required-features = ["serde", "scotch", "mpi"]

[package.metadata.example.parallel_mesh.templated-examples]
command = "mpirun -n {{NPROCESSES}}"

[[example]]
name = "parallel_io"
required-features = ["serde", "mpi"]

[package.metadata.example.parallel_io.templated-examples]
command = "mpirun -n 4"

[package.metadata.templated-examples]
NPROCESSES = ["2", "4"]
```

```
cargo templated-examples NPROCESSES 2,4
```

```
cargo run --example mixed_mesh --release
cargo run --example single_element_mesh --release
cargo run --example io --features "serde" --release
cargo mpirun -n 2 --example parallel_mesh --features "serde,scotch,mpi" --release
cargo mpirun -n 4 --example parallel_mesh --features "serde,scotch,mpi" --release
cargo mpirun -n 4 --example parallel_io --features "serde,mpi" --release
```

```
[[example]]
name = "io"
required-features = ["serde"]

[[example]]
name = "parallel_mesh"
required-features = ["serde", "scotch", "mpi"]

[package.metadata.example.parallel_mesh.templated-examples]
command = "mpirun -n {{NPROCESSES}}"

[[example]]
name = "parallel_io"
required-features = ["serde", "mpi"]

[package.metadata.example.parallel_io.templated-examples]
command = "mpirun -n 4"

[package.metadata.templated-examples]
NPROCESSES = ["2", "4"]
```



```
cargo templated-examples
```

Maybe one day...

```toml
[[example]]
name = "io"
required-features = ["serde"]

[[example]]
name = "parallel_mesh"
required-features = ["serde", "scotch", "mpi"]

[package.metadata.example.parallel_mesh.templated-examples]
command = "mpirun -n {{NPROCESSES}}"

[[example]]
name = "parallel_io"
required-features = ["serde", "mpi"]

[package.metadata.example.parallel_io.templated-examples]
command = "mpirun -n 4"
```

```toml
[[example]]
name = "io"
required-features = ["serde"]

[[example]]
name = "parallel_mesh"
required-features = ["serde", "scotch", "mpi"]
command = "mpirun -n {{NPROCESSES}}"

[[example]]
name = "parallel_io"
required-features = ["serde", "mpi"]
command = "mpirun -n 4"
```

Summary

- You can either:
  - put metadata in your Cargo.toml file
  - Add `//?` comments to your examples
- Then run all your examples with:

```
cargo templated-examples
```

- github.com/mscroggs/cargo-templated-examples

Thanks for listening!

Matthew Scroggs

Advanced Research Computing,
University College London

mscroggs.co.uk

matthew.scroggs.14@ucl.ac.uk

mscroggs

@mscroggs@mathstodon.xyz

@mscroggs.co.uk