The Fortran (community) standard library

Dr. José R. Alves Z.

TRANSVALOR

Why a stdlib for Fortran?

- Capitalize on common knowledge / experience / know-how

- Lowering the entry ticket for High Performance scientific development
  - For Python: NumPy, SciPy, Scikit-learn, Torch, etc
  - For C++: std, Boost, Eigen, etc.
  - For Fortran: !?

- Help the community avoid reinventing the wheel and focus on moving forward
  - Build on top of
  - Improve the existant

Agenda

- Avant-propos : Speaker
- Avant-propos : Fortran Ecosystem
- Stdlib’s Goals and Motivation
- Meta-programming (fypp)
- Features – Quick review
- Mini app demo
- Some lessons learned
- Contributing (Github)

Avant-propos : Speaker

TRANSVALOR

Speaker

José R. Alves Z.

Head of team and scientific developer at Transvalor S.A.
Multiphysics, deep learning, electromagnetism, damage-to-fracture modeling, parallel computing, numerical algorithm optimization

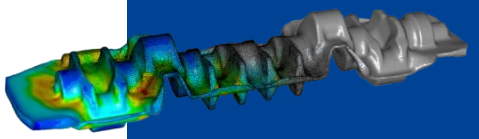
2016 - Ph.D on Simulation of magnetic pulse forming
CEMEF, MINES ParisTech – PSL University, France

2012 - Computational mechanics specialization
Full field FE modeling of glass fiber reinforced PP
CEMEF, MINES ParisTech – PSL University, France

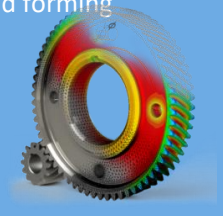
2011 - Mechanical engineer
Simon Bolivar University, Venezuela
Control Theory specialization at
Nagaoka University of Technology, Japan

Transvalor’s software solutions

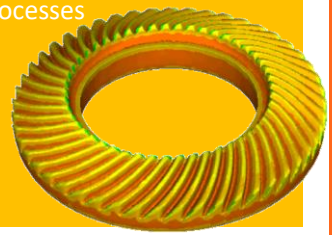
FORGE®
The reference in simulation for hot-warm-cold metal working



COLDFORM®
Dedicated simulation software for cold forming

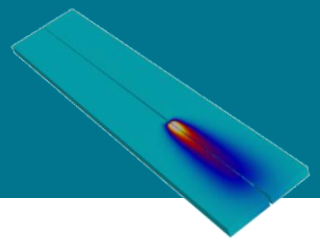


SIMHEAT®
Simulation solution for metal heat treatment processes

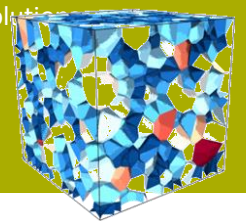


THERCAST®
Simulation software for metal casting processes

TRANSWELD®
Simulation software for welding processes

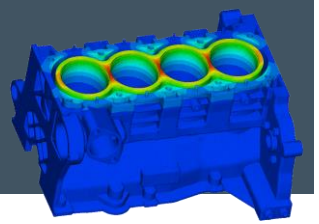


DIGIMU®
Solution for the prediction of microstructural evol



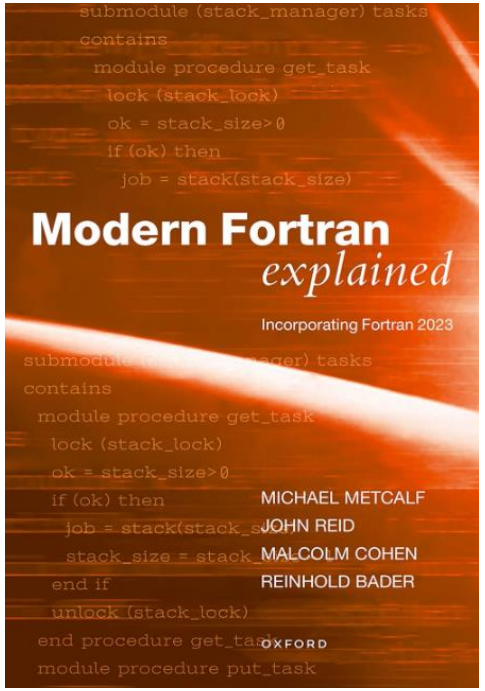
REM3D®
Simulation software for PU foams chemical foaming

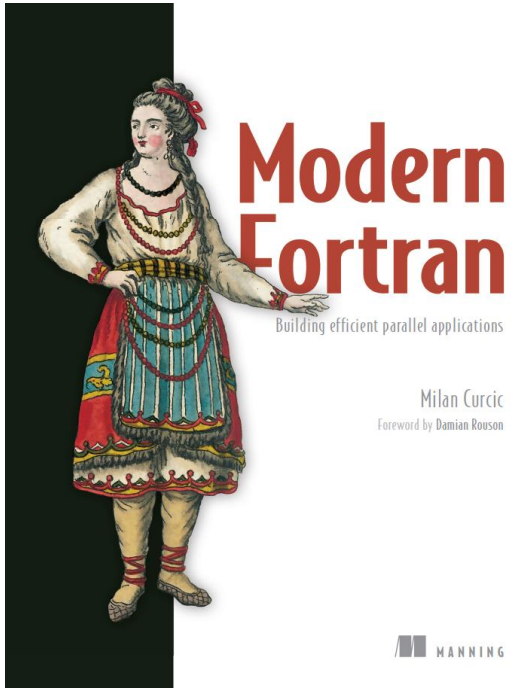
Z-set
Non-linear material & structural analysis suite



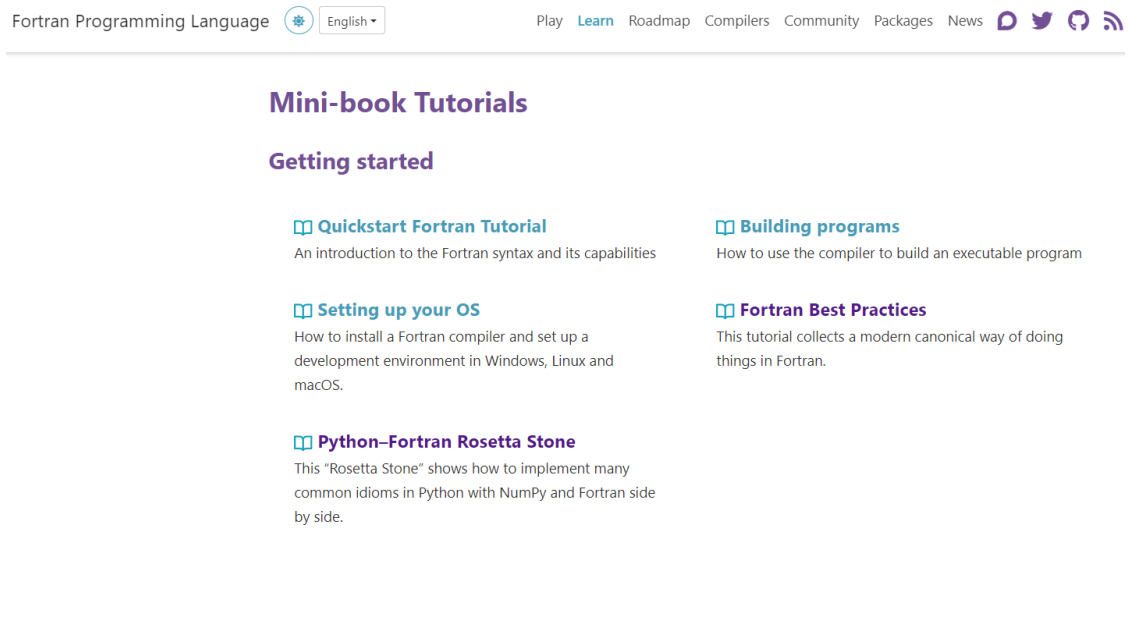
Avant-propos : Fortran & Ecosystem

Avant-propos / References





https://fortran-lang.org/learn/



cnrs

http://www.idris.fr/formations/fortran/

Leibniz-Rechenzentrum
der Bayerischen Akademie der Wissenschaften

Beratung

LRZ Dokumentationsplattform / ... / Lecture Notes and Materials

Programming with Fortran

https://doku.lrz.de/dyn/Doku_Kurse/Fortran/basics/Fortran_3days.pdf

Avant-propos / History

Free form / public/private
Derived Types
Allocatables / Pointers

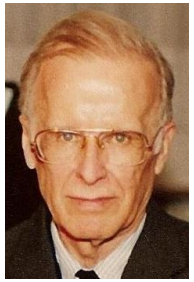
Type-bound procedures / abstract / extends
polymorphism
iso_c_binding (C interoperability)
move_alloc

Collective subroutines
teams / events

FORTRAN II
FORTRAN III
1958
FORTRAN 66
66
Fortran 90
91
Fortran 2003
2004
Fortran 2018
2018
Generics
202y

fortran-lang.org

1954/57
FORTRAN
John Backus @ IBM



61
FORTRAN IV

78
FORTRAN 77

Structured Programming
Implicit none

97
Fortran 95

2010
Fortran 2008

CoArray (native distributed parallelism)
do concurrent (native shared parallelism)
submodules
block construct
iso_fortran_env
contiguous

2023
Fortran 2023

Reduction in do concurrent

Milan Curcic
Ondrej Certik

https://fortran-lang.org/community/history/

Avant-propos / History

While Fortran being the oldest high-level programming language it was missing a solid open community for a long time

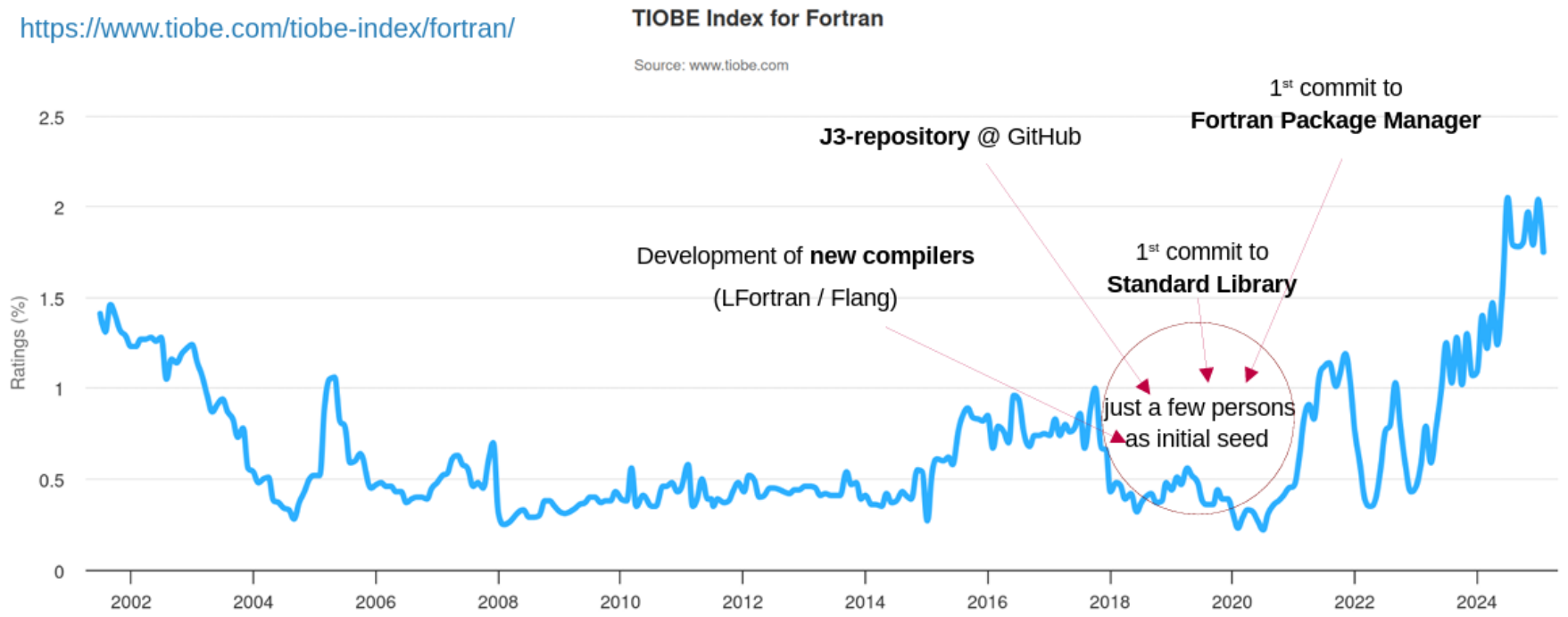
The notion of easy to use « Fortran » libraries was missing for a long time and most Fortranners had to re-invent the wheel

… the netlib-era-like development just don't hold anymore

« Research Software and its Developers: Random thoughts » by Balint Aradi
https://events.hifis.net/event/1741/contributions/12934/attachments/3354/7040/keynote_aradi.pdf

Building of communities: the revival of Fortran

https://www.tiobe.com/tiobe-index/fortran/



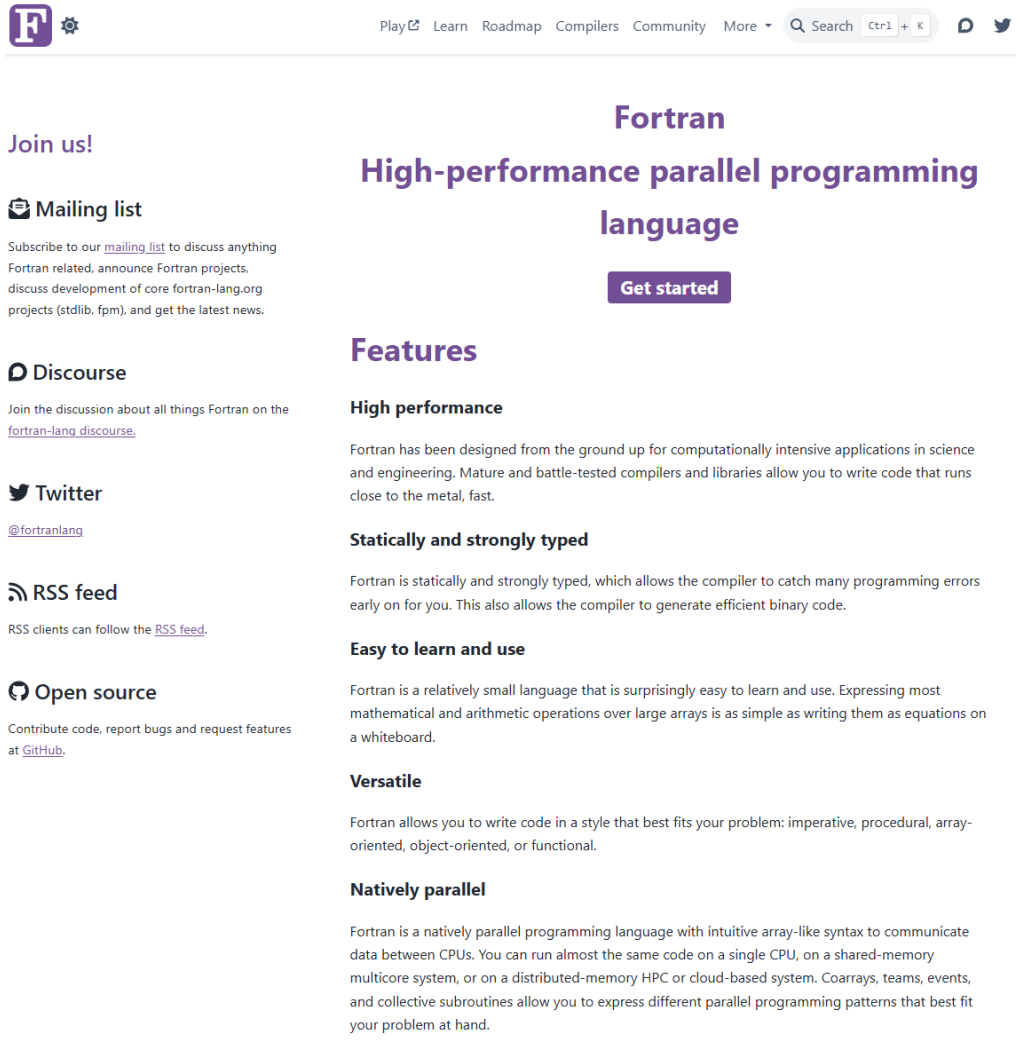
- Lowering entry barrier
- Making participation "cooler"
- Offering long-term perspective

- At which application level do we need a community?

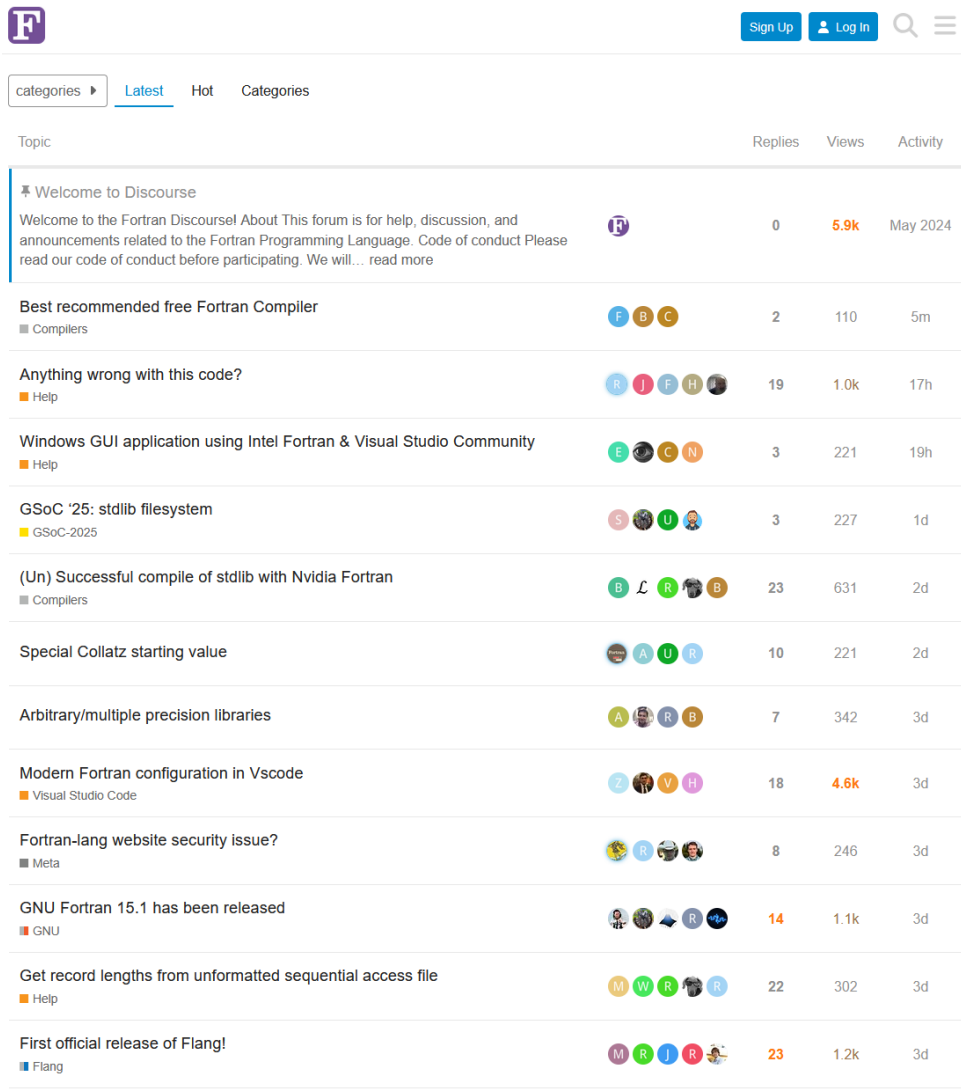
27

Fortran-lang

https://fortran-lang.org/



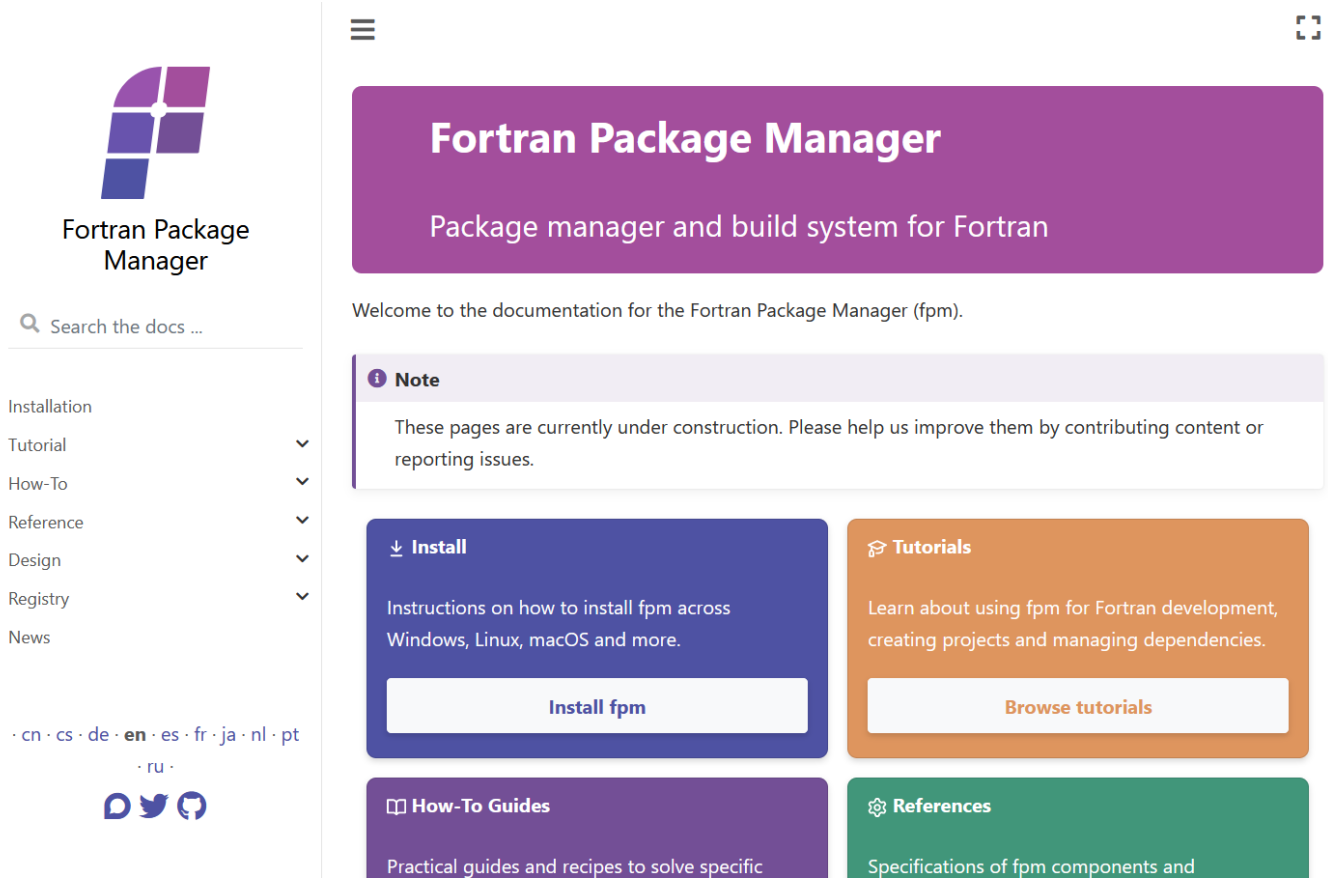
https://fortran-lang.discourse.group



TRANSVALOR

The Fortran Package Manager FPM

https://fpm.fortran-lang.org/



VS Code + Modern Fortran + fortls

https://code.visualstudio.com/
https://marketplace.visualstudio.com/items?itemName=fortran-lang.linter-gfortran

Key Features

- Syntax highlighting (Free and Fixed forms)
- Hover support, Signature help and Auto-completion
- GoTo/Peek implementation and Find/Peek references
- Project-wide and Document symbol detection and Renaming
- Native Language Server integration with `fortls`
- Linting support: GNU's `gfortran`, Intel's `ifort`, `ifx`, NAG's `nagfor`
- Interactive Debugger with UI
- Formatting with findent or fprettify
- Code snippets (more can be defined by the user see)

by Feb 2023 it had 270000 downloads
Today (June 13, 2025) it has more than 500000 installs

Online PlayGrounds

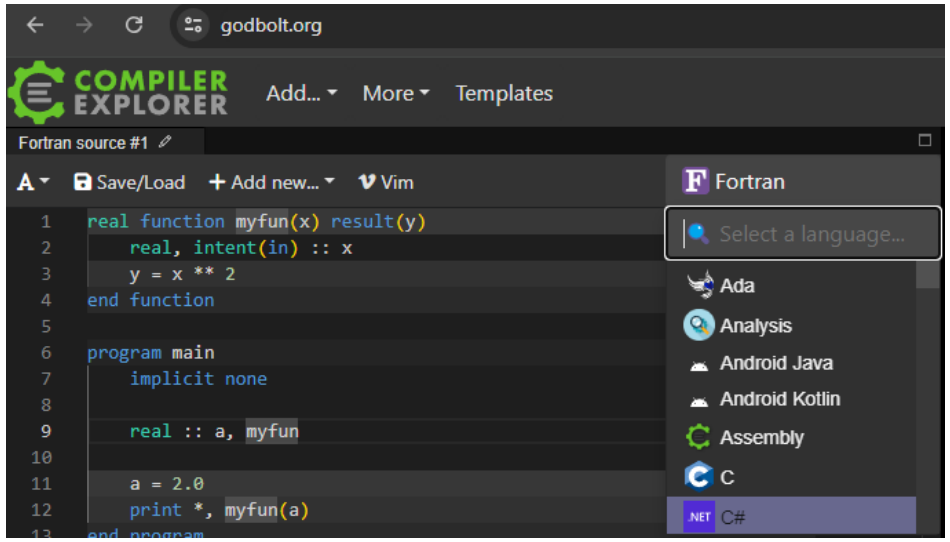
Name	URL	Characteristics
Fortran Play-Ground	https://play.fortran-lang.org/	From Fortran-lang community; GFortran compiler; stdlib standard library directly accessible
Compiler Explorer	https://godbolt.org/	Multi-language; possibility to compare execution results and generated assembly code generated by different compilers; sharing option...
tutorialspoint	https://www.tutorialspoint.com/	Multi-language with didactic material; GNU compiler; possibility to publicly share the code
LFortran	https://dev.lfortran.org/	Compiler in alpha version; graphical display; internal compiler representations and C++ generated code

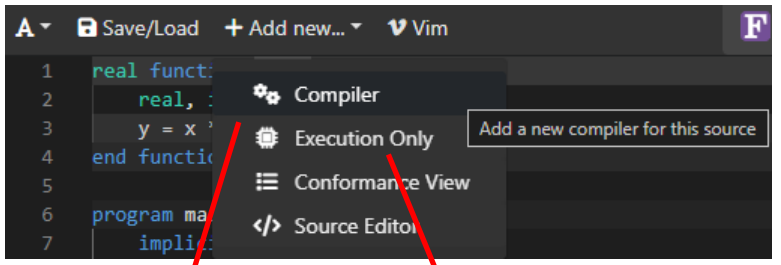
[V. Magin et al, Fortran… et puis quoi encore?]

C.E. enables easily

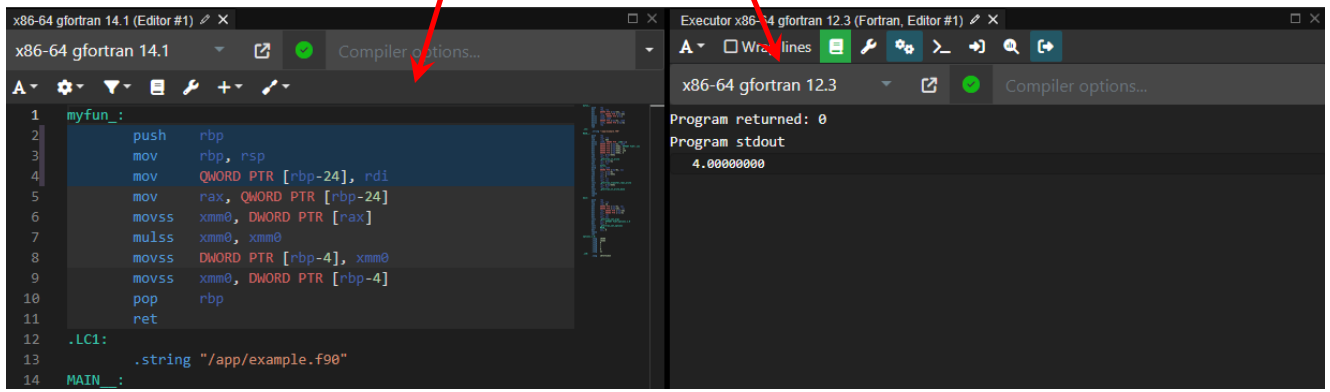
- Playing with code
- Understanding assembly code representation
- Comparing Compilers

Large selection of programming languages





Confortable multiple windows view with Compiler and Execution modes



Online PlayGrounds

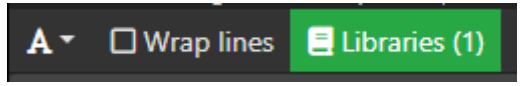
Name	URL	Characteristics
Fortran Play-Ground	https://play.fortran-lang.org/	From Fortran-lang community; GFortran compiler; stdlib standard library directly accessible
Compiler Explorer	https://godbolt.org/	Multi-language; possibility to compare execution results and generated assembly code generated by different compilers; sharing option...
tutorialspoint	https://www.tutorialspoint.com/	Multi-language with didactic material; GNU compiler; possibility to publicly share the code
LFortran	https://dev.lfortran.org/	Compiler in alpha version; graphical display; internal compiler representations and C++ generated code

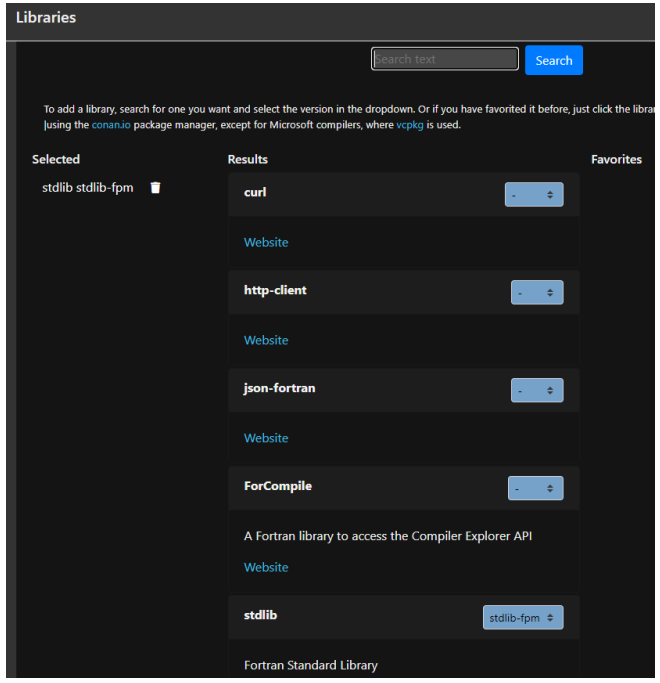
[V. Magin et al, Fortran… et puis quoi encore?]

C.E. enables easily

- Playing with code
- Understanding assembly code representation
- Comparing Compilers

Open source libraries for different languages





Stdlib’s Goals and Motivation

A library, A community

The Fortran Standard, as published by the ISO (https://wg5-fortran.org/), **does not have a Standard Library**. The goal of this project is to provide **a community driven and agreed upon de facto "standard" library for Fortran**, called a Fortran Standard Library (stdlib) (…)

Scope
The goal of the Fortran Standard Library is to achieve the following general scope:

- Utilities (containers, strings, files, OS/environment integration, unit testing & assertions, logging, ...)
- Algorithms (searching and sorting, merging, ...)
- Mathematics (linear algebra, sparse matrices, special functions, fast Fourier transform, random numbers, statistics, ordinary differential equations, numerical integration, optimization, ...)

https://github.com/fortran-lang/stdlib

A library, A community

The Fortran stdlib project has garnered over 1000 stars on GitHub!

jeremie.vandenplas Jun 2024

Great news: the Fortran stdlib 35 project has garnered over 1000 stars on GitHub!

Currently `stdlib` includes many utilities (hash maps 7 , strings 5 , IO 3 , logging 5 ,...), sorting procedures 4), and many mathematics-related procedures (linear algebra 9 , statistics 2 , random numbers 3),...).

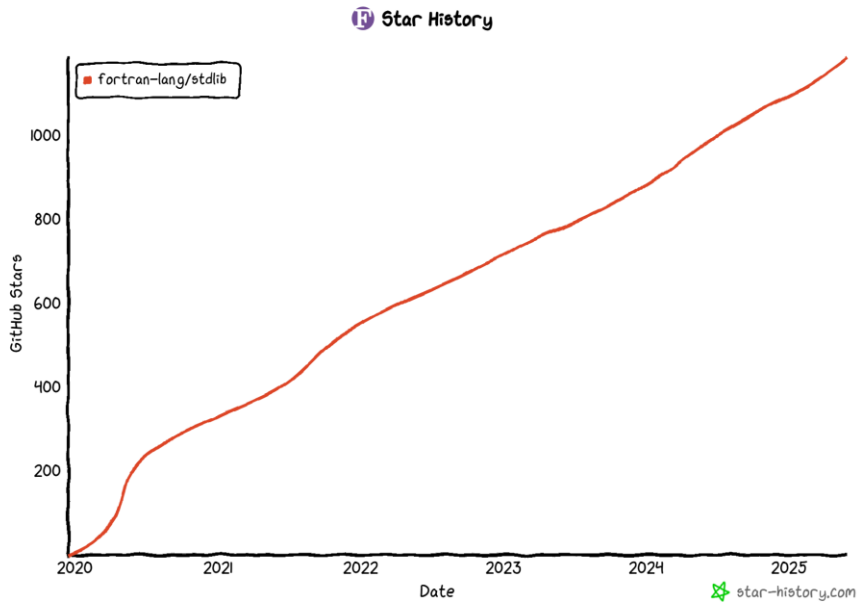
The largest and most recent addition to stdlib is indubiously the whole set of BLAS/LAPACK procedures, a project led by @FedericoPerini .

Thank you to all the contributors! I strongly believe that `stdlib` , in combination with fpm, makes Fortran even more accessible!

More to come soon!

Jun 2024

1 / 7
Jun 2024



Fortran stdlib: sufficient for scientific programming?

Help stdlib

This is the first time Bhanu has posted — let's welcome them to our community!

Bhanu 2 Nov 2024

I am very new to Fortran stdlib; however, I have been using C++ for my scientific programming since 2018. For fast analysis and a low program footprint, I wanted to switch to Fortran. As I am observing, LA algorithms are present with stdlib. Is the Fortran-stdlib sufficient for scientific programming, and can it replace LAPACK/BLAS?

Are there any detailed documents available for stdlib that have a stdlib - python/Matlab conversation table for the new Fortran programmer?

Some reflections about stdlib, where it stands and what it could become

loiseaujc 2 Mar 17

Although I have been familiar with `fpm` and `stdlib` for an extended period of time, I've started to take a keen interest in this discourse and its community only a year-ish ago. In all honesty, I have to say that it have had a transformative impact on my overall Fortran experience and re-ignited my hopes and desires of what a modern Fortran ecosystem for scientific computing could look like. There are so many people to thank for this that I will not even try to come up with an exhaustive list because I'll more likely that not forget too many of you still. You all know who you are and what we owe you anyways, so **thank you**.

Over the past year, `fpm` and `stdlib` have become absolutely critical components of my Fortran toolbox. So critical actually that, while I used to prototype pretty much everything with a mix-bag of `numpy` , `scipy` and possibly Julia until last spring, I now do so pretty much exclusively in Fortran directly. Trying a new idea has become as simple as `fpm new my_project` along with adding `stdlib = "*"` to the dependencies in the `toml` file and I'm good to go! This rejuvenated workflow is so simple and efficient that it largely contributed to my group and I being able to write new tools for our research that I wouldn't have phantom about even just a couple of years ago.

With the one year mark of my involvement in this community approaching, I think it is a good time to actually take some time and reflect on a number of things, including:

Mar 17

1 / 59
Mar 17

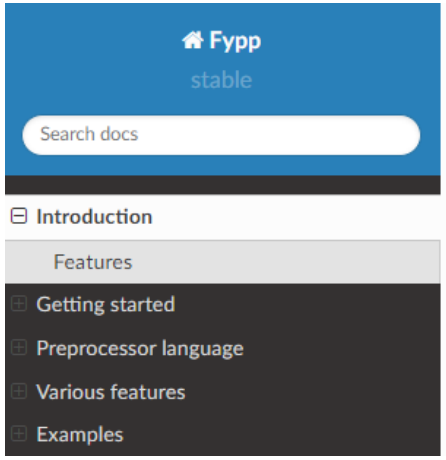
Mar 27

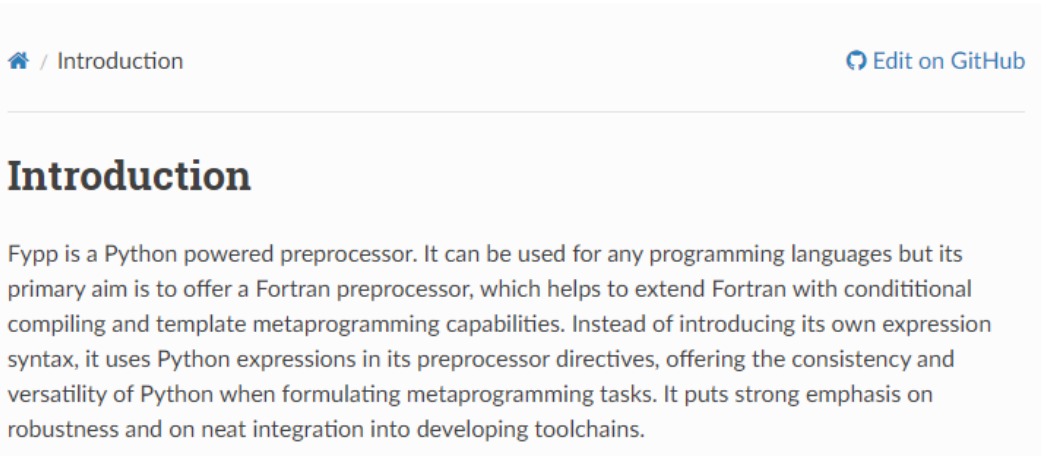
Meta-programming (FYPP)

TRANSVALOR

Generic & Metaprogramming with FYPP

https://fypp.readthedocs.io/en/stable/fypp.html#





Installing via conda

The last stable release of Fypp can be easily installed as conda package by issuing

```
conda install -c conda-forge fypp
```

Installing via pip

You can also use Pythons command line installer `pip` in order to download the stable release from the Fypp page on PyPI and install it on your system.

If you want to install Fypp into the module system of the active Python 3 interpreter (typically the case when you are using a Python virtual environment), issue

```
pip3 install fypp
```

Alternatively, you can install Fypp into the user space (under *~/.local*) with

```
pip3 install --user fypp
```

Running

The Fypp command line tool reads a file, preprocesses it and writes it to another file, so you would typically invoke it like:

```
fypp source.fpp source.f90
```

which would process *source.fpp* and write the result to *source.f90*. If input and output files are not specified, information is read from stdin and written to stdout.

CMake

One possible way of invoking the Fypp preprocessor within the CMake build framework is demonstrated below (thanks to Jacopo Chevallard for providing the very first version of this example):

```
### Pre-process: .fpp -> .f90 via Fypp

# Create a list of the files to be preprocessed
set(fppFiles file1.fpp file2.fpp file3.fpp)

# Pre-process
foreach(infileName IN LISTS fppFiles)

    # Generate output file name
    string(REGEX REPLACE ".fpp\$" ".f90" outfileName "${infileName}")

    # Create the full path for the new file
    set(outfile "${CMAKE_CURRENT_BINARY_DIR}/${outfileName}")

    # Generate input file name
    set(infile "${CMAKE_CURRENT_SOURCE_DIR}/${infileName}")

    # Custom command to do the processing
    add_custom-command(
        OUTPUT "${outfile}"
        COMMAND fypp "${infile}" "${outfile}"
        MAIN_DEPENDENCY "${infile}"
        VERBATIM)

    # Finally add output file to a list
    set(outFiles ${outFiles} "${outfile}")

endforeach(infileName)
```

Generic & Metaprogramming with FYPP

How FYPP is used in stdlib to facilitate generic development?

1. It is possible to create generic module interfaces for procedures which are type-kind-rank (TKR) generic

```
interface stdlib_sum
    !! version: experimental
    !!
    !!### Summary
    !! Sum elements of rank N arrays.
    !! ([Specification](../page/specs/stdlib_intrinsics.html#stdlib_sum))
    !!
    !!### Description
    !!
    !! This interface provides standard conforming call for sum of elements of
    !! The 1-D base implementation follows a chunked approach for optimizing pe
    !! The `N-D` interfaces calls upon the `(N-1)-D` implementation.
    !! Supported data types include `real`, `complex` and `integer`.
    !!
    #:for k, t, s in I_KINDS_TYPES + R_KINDS_TYPES + C_KINDS_TYPES
    pure module function stdlib_sum_1d_${s}$(a) result(s)
        ${t}$, intent(in) :: a(:)
        ${t}$ :: s
    end function
    pure module function stdlib_sum_1d_${s}$_mask(a,mask) result(s)
        ${t}$, intent(in) :: a(:)
        logical, intent(in) :: mask(:)
        ${t}$ :: s
    end function
    #:for rank in RANKS
    pure module function stdlib_sum_${rank}$d_${s}$( x, mask ) result( s )
        ${t}$, intent(in) :: x${ranksuffix(rank)}$
        logical, intent(in), optional :: mask${ranksuffix(rank)}$
        ${t}$ :: s
    end function
```

```
#:for k, t, s in I_KINDS_TYPES + R_KINDS_TYPES + C_KINDS_TYPES
pure module function stdlib_sum_1d_${s}$(a) result(s)
    ${t}$, intent(in) :: a(:)
    ${t}$ :: s
end function
```

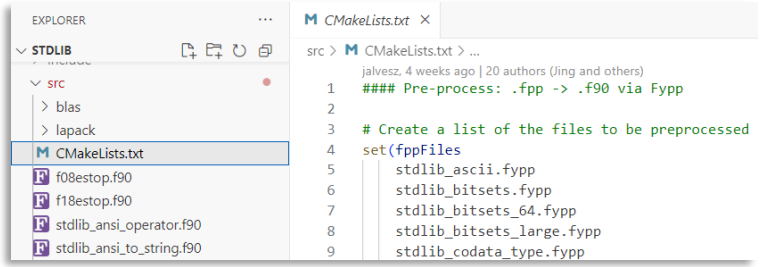
```
submodule(stdlib_intrinsics) stdlib_intrinsics_sum
    !! ([Specification](../page/specs/stdlib_intrinsics.html))
    use stdlib_kinds
    use stdlib_constants
    implicit none

    integer, parameter :: ilp = int64

contains

!================= 1D Base implementations ============
! This implementation is based on https://github.com/jalvesz/fast_math
#:for k, t, s in I_KINDS_TYPES + R_KINDS_TYPES + C_KINDS_TYPES
pure module function stdlib_sum_1d_${s}$(a) result(s)
    integer(ilp), parameter :: chunk = 64
    ${t}$, intent(in) :: a(:)
    ${t}$ :: s
    ${t}$ :: abatch(chunk)
    integer(ilp) :: i, n, r
    ! ----------------------------
    n = size(a,kind=ilp)
    r = mod(n,chunk)

    abatch(1:r)       = a(1:r)
    abatch(r+1:chunk) = zero_${s}$
    do i = r+1, n-r, chunk
     abatch(1:chunk) = abatch(1:chunk) + a(i:i+chunk-1)
    end do

    s = zero_${s}$
    do i = 1, chunk/2
      s = s + abatch(i)+abatch(chunk/2+i)
    end do
end function
```

2. Write down the implementation having parametrized the any or all TKR

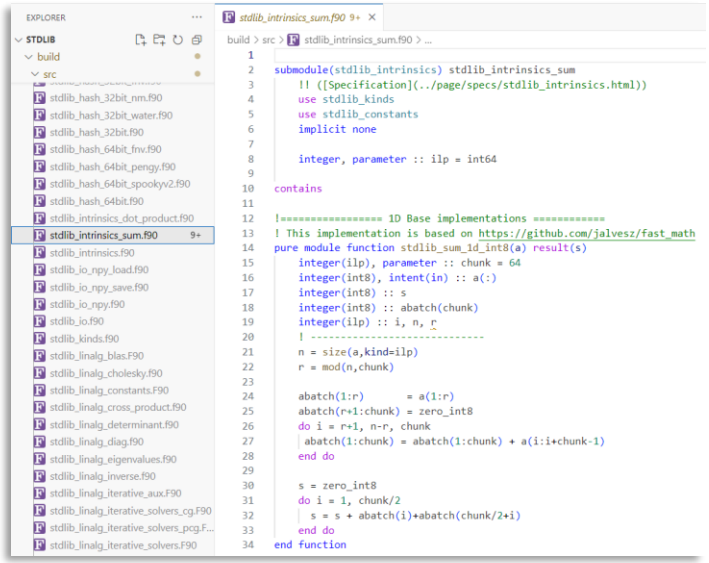
Generic & Metaprogramming with FYPP

How FYPP is used in stdlib to facilitate generic development?

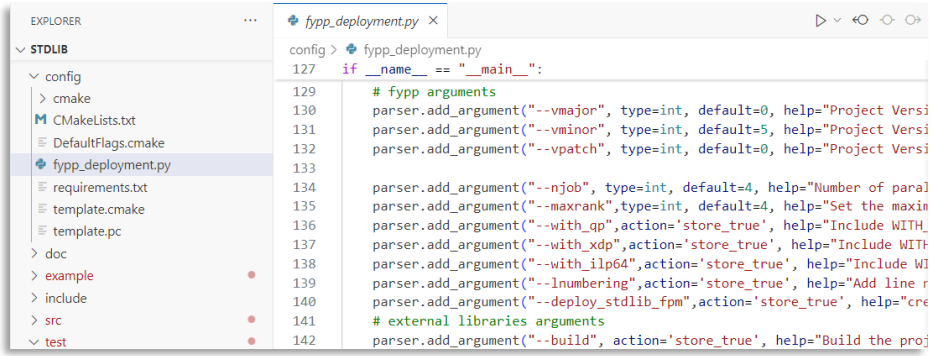
3. For CMake build, .fypp files can be added to a predefined collection of files to be preprocessed



4. during build, CMake will use fypp to create all .f90 files with all the versions in the build directory



3.bis. For fpm build, it does not natively support fypp BUT a python script facillity was added



4.bis. The fpm build includding fypp preprocessing can be invoked as

```
python config/fypp_deployment.py
fpm build --profile release
```

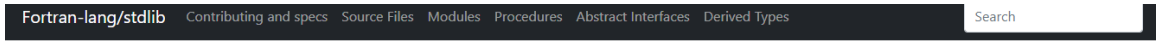
Interesting: the « --lnumbering » option enables to annotate the .f90 files such that compiler reporting points to the .fypp files instead.

Features – Quick Review

TRANSVALOR

The Fortran standard library: stdlib

Documentation : https://stdlib.fortran-lang.org/
Source Code : https://github.com/fortran-lang/stdlib



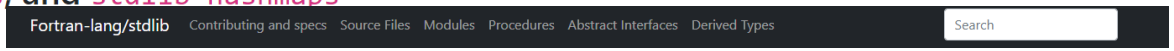
Hash maps

Contributing and specs / Specifications (specs) / Hash maps

Contributing and specs
Fortran stdlib License (MIT)
Changelog
Contributing
Contributor Code of Conduct
Style Guide
Workflow for Contributors
Specifications (specs)
terminal colors
array
ascii
bitsets
constants
error
hash
Hash maps
io

The `stdlib_hashmap_wrappers`, and `stdlib_hashmaps` modules

- The stdlib_hashmap_wrappers, and stdlib_hashmaps modules
  - Overview of hash maps
  - Licensing
  - The hash map modules
  - The stdlib_hashmap_wrappers module
    - The stdlib_hashmap_wrappers's constant, int_hash
    - The stdlib_hashmap_wrappers' module's derived types
    - Table of stdlib_hashmap_wrappers procedures
    - Specifications of the stdlib_hashmap_wrappers procedures
      - copy_key - Returns a copy of the key
        - Status
        - Description
        - Syntax
        - Class
        - Arguments
        - Example
      - copy_other - Returns a copy of the other data
        - Status
        - Description
        - Syntax
        - Class
        - Arguments
        - Example
      - fibonacci_hash - maps an integer to a smaller number of bit
        - Status
        - Description
      - fnv_1_hasher- calculates a hash code from a key
        - Status
        - Description
        - Syntax
        - Class
        - Argument
        - Result character
        - Result value
        - Note



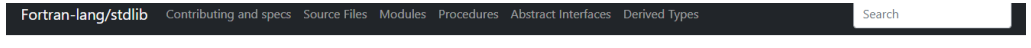
linalg

Contributing and specs / Specifications (specs) / linalg

Contributing and specs
Fortran stdlib License (MIT)
Changelog
Contributing
Contributor Code of Conduct
Style Guide
Workflow for Contributors
Specifications (specs)
terminal colors
array
ascii
bitsets
constants
error
hash
Hash maps
io

Linear Algebra

- Linear Algebra
  - BLAS and LAPACK
    - Status
    - Description
    - Syntax
    - Example
    - Licensing
  - diag - Create a diagonal array or extract the diagonal elements
    - Status
    - Description
    - Syntax
    - Arguments
    - Return value
    - Example
  - eye - Construct the identity matrix
    - Status
    - Class
    - Description
    - Syntax
    - Arguments
    - Return value
      - Warning
    - Example
  - trace - Trace of a matrix
    - Status
    - Description
    - Syntax
    - Arguments
    - Return value
    - Example
  - outer_product - Computes the outer product of two vectors
    - Status
    - Description
    - Syntax
    - Arguments
    - Return value



sorting

Contributing and specs / Specifications (specs) / sorting

Contributing and specs
Fortran stdlib License (MIT)
Changelog
Contributing
Contributor Code of Conduct
Style Guide
Workflow for Contributors
Specifications (specs)
terminal colors
array
ascii
bitsets
constants
error
hash
Hash maps
io

The `stdlib_sorting` module

- The stdlib_sorting module
  - Overview of sorting
  - Overview of the module
    - The parameters int_index and int_index_low
    - The module subroutines
      - Licensing
      - The ORD_SORT subroutine
      - The SORT_INDEX subroutine
      - The SORT subroutine
      - The RADIX_SORT subroutine
    - Specifications of the stdlib_sorting procedures
      - ord_sort - sorts an input array
        - Status
        - Description
        - Syntax
        - Class
        - Arguments
        - Notes
        - Example
      - sort - sorts an input array
        - Status
        - Description
        - Syntax
        - Class
        - Arguments
        - Notes
        - Example
      - radix_sort - sorts an input array
        - Status
        - Description
        - Syntax
        - Class
        - Arguments
        - Notes
        - Example
      - sort_index - creates an array of sorting indices for an input array, while also sorting the array.
        - Status
        - Description

The Fortran standard library: stdlib

Specifications (specs)

terminal colors
array
ascii
bitsets
constants
error
state_type
hash
Hash maps
intrinsics
io
kinds
linalg
linalg_state_type
logger
math
optval
quadrature
random
selection
sorting
sparse
specialfunctions
specialfunctions_activations
specialfunctions_gamma
stats
stats_distribution_exponential
stats_distribution_normal
stats_distribution_uniform
str2num
string_type
stringlist_type
strings
system
version

Stdlib ansi

`ansi_code` type

The `ansi_code` type represent an ANSI escape sequence with a style, foreground color and background color attribute. By default the instances of this type are empty and represent no escape sequence.

Status

Experimental

Example

```
program example_ansi_color
  use stdlib_ansi, only : fg_color_blue, style_bold, style_reset, ansi_code, &
    & operator(//), operator(+)
  implicit none
  type(ansi_code) :: highlight, reset

  print '(a)', highlight // "Dull text message" // reset

  highlight = fg_color_blue + style_bold
  reset = style_reset

  print '(a)', highlight // "Colorful text message" // reset
end program example_ansi_color
```

The `stdlib_ansi` module

Stdlib io

`loadtxt` - load a 2D array from a text file

Status

Experimental

Description

Loads a rank-2 `array` from a text file.

Syntax

```
call loadtxt (filename, array [, skiprows] [, max_rows] [, fmt] [, delimiter])
```

`savetxt` - save a 2D array into a text file

Status

Experimental

Description

Saves a rank-2 `array` into a text file.

Syntax

```
call savetxt (filename, array [, delimiter])
```

`load_npy`

Status

Experimental

Description

Loads an `array` from a npy formatted binary file.

Syntax

```
call load_npy (filename, array[, iostat][, iomsg])
```

`save_npy`

Status

Experimental

Description

Saves an `array` into a npy formatted binary file.

Syntax

```
call save_npy (filename, array[, iostat][, iomsg])
```

Mirror facilities to:

numpy.load
numpy.save

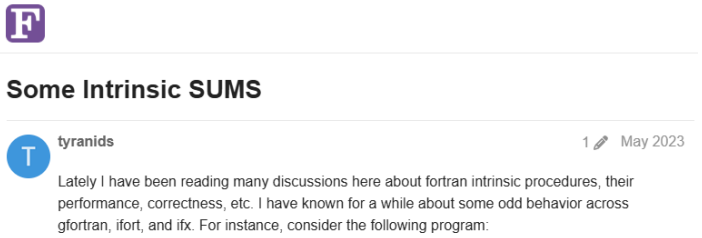
numpy.loadtxt
numpy.savetxt

Note: .npy is the numpy native binary format for array exchange

Warning: 2D arrays should be transposed if passing data between Python and Fortran

Stdlib intrinsics

A debate about accuracy of basic procedures …

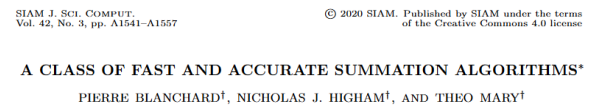


Some Intrinsic SUMS

tyranids — 1 — May 2023

Lately I have been reading many discussions here about fortran intrinsic procedures, their performance, correctness, etc. I have known for a while about some odd behavior across gfortran, ifort, and ifx. For instance, consider the following program:

Inspired by: https://epubs.siam.org/doi/epdf/10.1137/19M1257780
a fast and accurate implementation was proposed as alternative to the compiler intrinsics



SIAM J. SCI. COMPUT.
Vol. 42, No. 3, pp. A1541–A1557

© 2020 SIAM. Published by SIAM under the terms of the Creative Commons 4.0 license

A CLASS OF FAST AND ACCURATE SUMMATION ALGORITHMS*

PIERRE BLANCHARD†, NICHOLAS J. HIGHAM†, AND THEO MARY†

```
n = size(a,kind=ilp)
r = mod(n,chunk)

abatch(1:r)       = a(1:r)
abatch(r+1:chunk) = zero_int8
do i = r+1, n-r, chunk
 abatch(1:chunk) = abatch(1:chunk) + a(i:i+chunk-1)
end do

s = zero_int8
do i = 1, chunk/2
  s = s + abatch(i)+abatch(chunk/2+i)
end do
```

sum

sum r32	[ns/eval]	Speed-Up	relative error
intrinsic	1.2100	1.00	3.3794E-06
kahan	0.1800	6.72	1.0425E-07
chunk	0.1100	11.00	1.1265E-07

sum r64	[ns/eval]	Speed-Up	relative error
intrinsic	1.3000	1.00	5.9269E-15
kahan	0.3100	4.19	1.7286E-16
chunk	0.1500	8.67	2.1416E-16

sum r32 mask	[ns/eval]	Speed-Up	relative error
intrinsic	4.1250	1.00	1.5687E-06
kahan	0.1600	25.78	9.1493E-08
chunk	0.1600	25.78	8.8453E-08

sum r64 mask	[ns/eval]	Speed-Up	relative error
intrinsic	4.0350	1.00	2.9428E-15
kahan	0.3750	10.76	1.2179E-16
chunk	0.2450	16.47	1.2768E-16

dot_product

dot r32	[ns/eval]	Speed-Up	relative error
intrinsic	1.0600	1.00	3.2735E-06
kahan	0.1500	7.07	9.8348E-08
chunk	0.1000	10.60	1.1587E-07

dot r64	[ns/eval]	Speed-Up	relative error
intrinsic	1.2100	1.00	5.8091E-15
kahan	0.3300	3.67	1.8407E-16
chunk	0.2000	6.05	2.0528E-16

Stdlib intrinsics

… thanks to such debate, stdlib now proposes these reference implementations :

Efficiency – Accuracy - A reference community version

stdlib_sum

Syntax

```
res = stdlib_sum (x [,mask] )

res = stdlib_sum (x, dim [,mask] )
```

stdlib_dot_product

Syntax

```
res = stdlib_dot_product (x, y)
```

stdlib_sum_kahan

Syntax

```
res = stdlib_sum_kahan (x [,mask] )

res = stdlib_sum_kahan (x, dim [,mask] )
```

stdlib_dot_product_kahan

Syntax

```
res = stdlib_dot_product_kahan (x, y)
```

```
elemental subroutine kahan_kernel_<kind>(a,s,c)
    type(<kind>), intent(in) :: a
    type(<kind>), intent(inout) :: s
    type(<kind>), intent(inout) :: c
    type(<kind>) :: t, y
    y = a - c
    t = s + y
    c = (t - s) - y
    s = t
end subroutine
```

Stdlib linalg

BLAS/LAPACK

Reference-LAPACK 3.10.1

- Stdlib used fypp metaprograming to extend for single, double, extended and quadruple precission
- Common interfaces for all precissions

```
!> AXPY: constant times a vector plus a vector.
interface axpy
    module procedure stdlib_saxpy
    module procedure stdlib_daxpy
    module procedure stdlib_qaxpy
    module procedure stdlib_caxpy
    module procedure stdlib_zaxpy
    module procedure stdlib_waxpy
end interface axpy
```

Rename the original functions to respect LAPACK's licence conditions
Use a kind-agnostic name for the generic interface

- C-Preprocessing macros for linking against optimized libraries

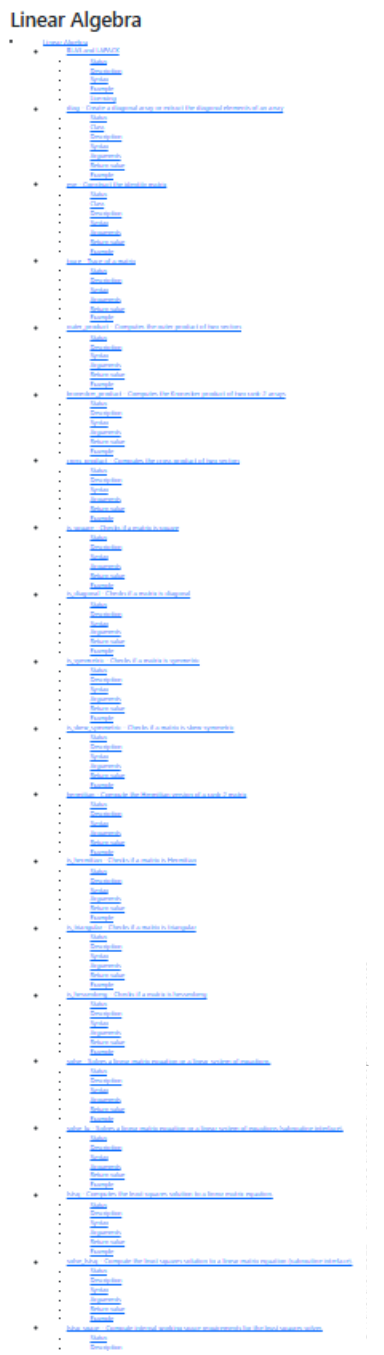
```
# Link against appropriate external BLAS and LAPACK libraries, if necessary
[build]
link = ["blas", "lapack"]

[dependencies]
stdlib="*"

# Macros are only needed if using an external library
[preprocess]
[preprocess.cpp]
macros = ["STDLIB_EXTERNAL_BLAS", "STDLIB_EXTERNAL_LAPACK"]
```

or directly via compiler flags:

```
fpm build --flag "-DSTDLIB_EXTERNAL_BLAS -DSTDLIB_EXTERNAL_LAPACK -lblas -llapack".
```

Macros will enable wrapping single precission (sp) and double precission (dp) procedures.

For extended (xdp) and quadruple (qp) precissions the stdlib versions will remain as the fallback option.



Stdlib linalg

High Level APIs

`solve` - Solves a linear matrix equation or a linear system of equations.

Status

Stable

Description

This function computes the solution to a linear matrix equation $A \cdot x = b$, where A is a square, full-rank, `real` or `complex` matrix.

Result vector or array `x` returns the exact solution to within numerical precision, provided that the matrix is not ill-conditioned. An error is returned if the matrix is rank-deficient or singular to working precision. The solver is based on LAPACK's `*GESV` backends.

Syntax

`Pure` interface:

```
x = solve (a, b)
```

Expert interface:

```
x = solve (a, b [, overwrite_a], err)
```

`det` - Computes the determinant of a square matrix

Status

Stable

Description

This function computes the determinant of a `real` or `complex` square matrix.

This interface comes with a `pure` version `det(a)`, and a non-pure version `det(a,overwrite_a,err)` that allows for more expert control.

Syntax

```
c = det (a [, overwrite_a, err])
```

`schur` - Compute the Schur decomposition of a matrix

Status

Experimental

Description

This subroutine computes the Schur decomposition of a `real` or `complex` matrix: $A = ZTZ^H$, where Z is unitary (orthonormal) and T is upper-triangular (for complex matrices) or quasi-upper-triangular (for real matrices, with possible 2×2 blocks on the diagonal). Matrix A has size `[n,n]`.

The results are returned in output matrices T and Z. Matrix T is the Schur form, and matrix Z is the unitary transformation matrix such that $A = ZTZ^H$. If requested, the eigenvalues of T can also be returned as a `complex` array of size `[n]`.

Syntax

```
call schur (a, t [, z,] [, eigvals] [, overwrite_a] [, storage] [, err])
```

`norm` - Computes the vector norm of a generic-rank array.

Status

Experimental

Description

This function computes one of several vector norms of `real` or `complex` array A, depending on the value of the `order` input argument. A may be an array of any rank.

Syntax

```
x = norm (a, order, [, dim, err])
```

Arguments

`a`: Shall be a rank-n `real` or `complex` array containing the data. It is an `intent(in)` argument.

`order`: Shall be an `integer` value or a `character` flag that specifies the norm type, as follows. It is an `intent(in)` argument.

Integer input	Character Input	Norm type
`-huge(0)`	`'-inf', '-Inf'`	Minimum absolute value (\min_i{ \left
`1`	`'1'`	1-norm (\sum_i{ \left
`2`	`'2'`	Euclidean norm $\sqrt{\sum_i a_i^2}$
`>=3`	`'3','4',...`	p-norm (\left(\sum_i{ \left
`huge(0)`	`'inf', 'Inf'`	Maximum absolute value (\max_i{ \left

`dim` (optional): Shall be a scalar `integer` value with a value in the range from `1` to `n`, where `n` is the rank of the array. It is an `intent(in)` argument.

... and several more ...

Stdlib linalg

```fortran
program example_norm
  use stdlib_linalg, only: norm, linalg_state_type
  implicit none

  real :: a(3,3),na
  integer :: j
  type(linalg_state_type) :: err

  ! a = [   -3.00000000   0.00000000   3.00000000
  !         -2.00000000   1.00000000   4.00000000
  !         -1.00000000   2.00000000   5.00000000   ]
  a = reshape([(j-4,j=1,9)], [3,3])

  print "(' a = [   ',3(g0,3x),2(/9x,3(g0,3x)),']')", transpose(a)

  print *, 'Euclidean norm      = ',norm(a, 2)
  print *, 'Euclidean norm      = ',norm(a, '2')
  print *, 'Rows norms          = ',norm(a, 2, dim=2)
  print *, 'Columns norms       = ',norm(a, 2, dim=1)

  ! Infinity norms
  print *, 'maxval(||a||)       = ',norm(a, 'inf')
  print *, 'maxval(||a(i,:)||)  = ',norm(a, 'inf', dim=2)
  print *, 'minval(||a||)       = ',norm(a, '-inf')
  print *, 'minval(||a(:,i)||)  = ',norm(a, '-inf', dim=1)

  ! Catch Error:
  print *, 'invalid: ',norm(a,'inf', dim=4, err=err)
  print *, 'error =  ',err%print()

end program example_norm
```

x86-64 gfortran 13.2 | Compiler options...

```
Program returned: 0
Program stdout
 a = [   -3.00000000   0.00000000   3.00000000
         -2.00000000   1.00000000   4.00000000
         -1.00000000   2.00000000   5.00000000   ]
 Euclidean norm      =    8.30662346
 Euclidean norm      =    8.30662346
 Rows norms          =    4.24264050       4.58257580       5.47722578
 Columns norms       =    3.74165750       2.23606801       7.07106781
 maxval(||a||)       =    5.00000000
 maxval(||a(i,:)||)  =    3.00000000       4.00000000       5.00000000
 minval(||a||)       =    0.00000000
 minval(||a(:,i)||)  =    1.00000000       0.00000000       3.00000000
 invalid:    0.00000000       0.00000000       0.00000000
 error =  [norm] returned Value Error: dimension 4 is out of rank for shape(a)= [3 3 3]
```

Stdlib sorting

The module subroutines

The `stdlib_sorting` module provides three different overloaded subroutines intended to sort three different kinds of arrays of data:

- `ORD_SORT` is intended to sort simple arrays of intrinsic data that have significant sections that were partially ordered before the sort;
- `SORT_INDEX` is based on `ORD_SORT`, but in addition to sorting the input array, it returns indices that map the original array to its sorted version. This enables related arrays to be re-ordered in the same way;
- `SORT` is intended to sort simple arrays of intrinsic data that are effectively unordered before the sort;
- `RADIX_SORT` is intended to sort fixed width intrinsic data types (integers and reals).

`ord_sort` - sorts an input array

Status

Experimental

Description

Returns an input `array` with the elements sorted in order of increasing, or decreasing, value.

Syntax

```
call ord_sort ( array[, work, reverse ] )
```

`radix_sort` - sorts an input array

Status

Experimental

Description

Returns an input array with the elements sorted in order of increasing, or decreasing, value.

Syntax

```
call radix_sort ( array[, work, reverse] )
```

`sort` - sorts an input array

Status

Experimental

Description

Returns an input array with the elements sorted in order of increasing, or decreasing, value.

Syntax

```
call sort ( array[, reverse] )
```

`sort_index` - creates an array of sorting indices for an input array, while also sorting the array.

Status

Experimental

Description

Returns the input `array` sorted in the direction requested while retaining order stability, and an integer array whose elements would sort the input `array` to produce the output `array`.

Syntax

```
call sort_index ( array, index[, work, iwork, reverse ] )
```

Stdlib sparse

5 supported formats

COO: The COOrdinates compressed sparse format

CSR : The Compressed Sparse Row or Yale format

CSC : The Compressed Sparse Column format

ELLPACK : ELL-pack storage format

SELL-C : The Sliced ELLPACK with Constant blocks format
[Kreutzer M. et al, *"A unified sparse matrix data format for modern processors with wide simd units"*, 2013]

Data accessors

`add`- sparse matrix data accessors

Status

Experimental

Description

Type-bound procedures to enable adding data in a sparse matrix.

Syntax

`call matrix%add(i,j,v)` or `call matrix%add(i(:),j(:),v(:,:))`

`at`- sparse matrix data accessors

Status

Experimental

Description

Type-bound procedures to enable requesting data from a sparse matrix.

Syntax

```
v = matrix%at(i,j)
```

Stdlib sparse

from_ijv

`call from_ijv (sparse,row,col[,data,nrows,ncols,num_nz_rows,chunk])`

Arguments

`sparse` : Shall be a `COO`, `CSR`, `ELL` or `SELLC` type. The graph object will be returned with a canonical shape after sorting and removing duplicates from the `(row,col,data)` triplet. If the graph is `COO_type` no data buffer is allowed. It is an `intent(inout)` argument.

`row` : rows index array. It is an `intent(in)` argument.

`col` : columns index array. It is an `intent(in)` argument.

`data`, `optional`: `real` or `complex` data array. It is an `intent(in)` argument.

`nrows`, `optional`: number of rows, if not given it will be computed from the `row` array. It is an `intent(in)` argument.

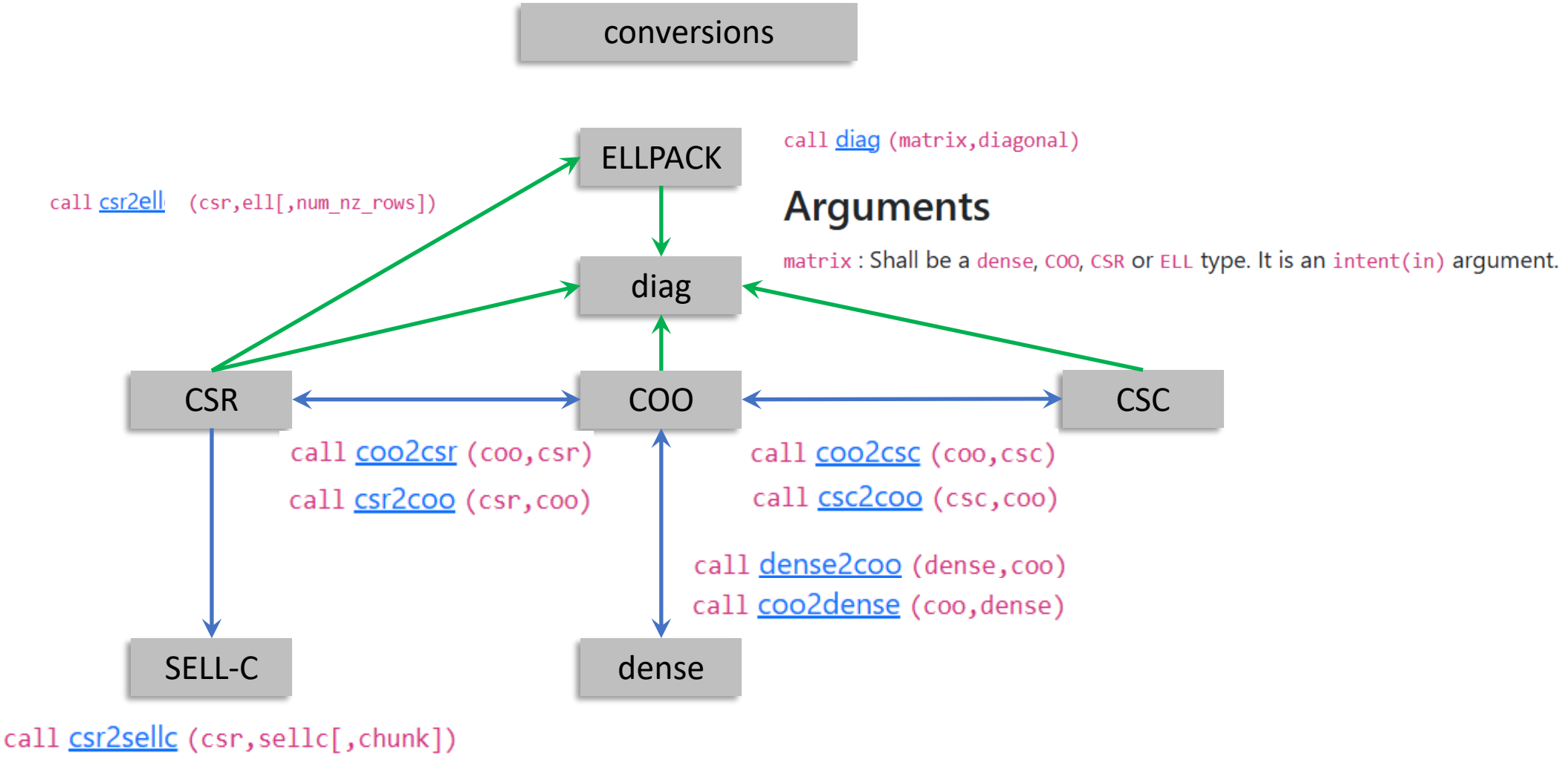
`ncols`, `optional`: number of columns, if not given it will be computed from the `col` array. It is an `intent(in)` argument.

`num_nz_rows`, `optional`: number of non zeros per row, only valid in the case of an `ELL` matrix, by default it will computed from the largest row. It is an `intent(in)` argument.

`chunk`, `optional`: chunk size, only valid in the case of a `SELLC` matrix, by default it will be taken from the `SELLC` default attribute chunk size. It is an `intent(in)` argument.

A facility procedure to initialize a sparse matrice from row, column, values triplets

Stdlib sparse



Stdlib sparse

SpMV Kernel

`spmv` - Sparse Matrix-Vector product

Status

Experimental

Description

Provide sparse matrix-vector product kernels for the current supported sparse matrix types.

$$y = \alpha * op(M) * x + \beta * y$$

Syntax

`call spmv (matrix,vec_x,vec_y [,alpha,beta,op])`

```
program example_sparse_spmv
    use stdlib_linalg_constants, only: dp
    use stdlib_sparse
    implicit none

    integer, parameter :: m = 4, n = 2
    real(dp) :: A(m,n), x(n)
    real(dp) :: y_dense(m), y_coo(m), y_csr(m)
    real(dp) :: alpha, beta
    type(COO_dp_type) :: COO
    type(CSR_dp_type) :: CSR

    call random_number(A)
    ! Convert from dense to COO and CSR matrices
    call dense2coo( A , COO )
    call coo2csr( COO , CSR )

    ! Initialize vectors
    x       = 1._dp
    y_dense = 2._dp
    y_coo   = y_dense
    y_csr   = y_dense

    ! Perform matrix-vector product
    alpha = 3._dp; beta = 2._dp
    y_dense = alpha * matmul(A,x) + beta * y_dense
    call spmv( COO , x , y_coo , alpha = alpha, beta = beta )
    call spmv( CSR , x , y_csr , alpha = alpha, beta = beta )

    print *, 'dense :', y_dense
    print *, 'coo   :', y_coo
    print *, 'csr   :', y_csr

end program example_sparse_spmv
```

Stdlib str2num

`to_num` - conversion of strings to numbers

Status

Experimental

Description

Convert a string or an array of strings to numerical types.

Syntax

```
number = to_num (string, mold)
```

`to_num_from_stream` - conversion of a stream of values in a string to numbers

Status

Experimental

Description

Convert a stream of values in a string to an array of values.

Syntax

```
number = to_num_from_stream (string, mold)
```

Stdlib str2num

```fortran
program example_string_to_number
    use stdlib_kinds, only: dp
    use stdlib_str2num, only: to_num
    implicit none

    block
        character(:), allocatable :: txt
        real(dp) :: x
        txt = ' 8.8541878128e-12 '
        x = to_num( txt , x )
        print *, x
    end block

    block
        character(24) :: txt(5)
        real(dp) :: x(5)
        txt = '8.8541878128e-12'
        x = to_num( txt , x )
        print *, x
    end block
end program example_string_to_number
```

```fortran
program example_stream_of_strings_to_numbers
    use stdlib_kinds, only: dp
    use stdlib_str2num, only: to_num_from_stream
    implicit none
    character(:), allocatable, target :: chain
    character(len=:), pointer :: cptr
    real(dp), allocatable :: r(:), p(:)
    integer :: i

    chain = " 1.234   1.E1 1e0     0.1234E0  12.21e+001 -34.5E1"
    allocate( r(6), p(6) )
    !> Example for streamline conversion using `to_num_from_stream`
    cptr => chain
    do i =1, 6
        !> the pointer is shifted within the function
        r(i) = to_num_from_stream( cptr , r(i) )
    end do
    read(chain,*) p
    print *, "Reading with to_num_from_stream"
    print *, r
    print *, "Reading with formatted read"
    print *, p

end program example_stream_of_strings_to_numbers
```

- An **elemental** interface to convert whole arrays at once or
- A **pointer** function interface to convert « **streams** of characters » …
  Ideal for reading formatted ASCII files

Stdlib system

`run` - Execute an external process synchronously

Status

Experimental

Description

The `run` interface allows execution of external processes using a single command string or a list of arguments.
Processes run synchronously, meaning execution is blocked until the process finishes.
Optional arguments enable the collection of standard output and error streams, as well as sending input via standard input. Additionally, a callback function can be specified to execute upon process completion, optionally receiving a user-defined payload.

Syntax

```
process = stdlib_system (args [, stdin] [, want_stdout] [, want_stderr] [, callback] [, payload])
```

`runasync` - Execute an external process asynchronously

Status

Experimental

Description

The `runasync` interface allows execution of external processes using a single command string or a list of arguments.
Processes are run asynchronously (non-blocking), meaning execution does not wait for the process to finish.
Optional arguments enable the collection of standard output and error streams, as well as sending input via standard input. Additionally, a callback function can be specified to execute upon process completion, optionally receiving a user-defined payload.

Syntax

```
process = stdlib_system (args [, stdin] [, want_stdout] [, want_stderr] [, callback] [, payload])
```

`is_running` - Check if a process is still running

Status

Experimental

Description

The `is_running` interface provides a method to check if an external process is still running.
This is useful for monitoring the status of asynchronous processes created with the `run` interface.

Syntax

```
status = stdlib_system (process)
```

`is_completed` - Check if a process has completed execution

Status

Experimental

Description

The `is_completed` interface provides a method to check if an external process has finished execution.
This is useful for determining whether asynchronous processes created with the `run` interface have terminated.

Syntax

```
status = stdlib_system (process)
```

`elapsed` - Return process lifetime in seconds

Status

Experimental

Description

The `elapsed` interface provides a method to calculate the total time that has elapsed since a process was started.
This is useful for tracking the duration of an external process or for performance monitoring purposes.

The result is a real value representing the elapsed time in seconds, measured from the time the process was created.

Syntax

```
delta_t = stdlib_system (process)
```

`kill` - Terminate a running process

Status

Experimental

Description

The `kill` interface is used to terminate a running external process. It attempts to stop the process and returns a boolean flag indicating whether the operation was successful. This interface is useful when a process needs to be forcefully stopped, for example, if it becomes unresponsive or if its execution is no longer required.

Syntax

```
call stdlib_system (process, success)
```

System and sub-processing module

The `stdlib_system` module provides interface for interacting with external processes, enabling the execution and monitoring of system commands or applications directly from Fortran.

Stdlib system

Comming Up … GSoC 2025

GSoC ‘25: stdlib filesystem

Projects GSoC-2025

suprit05 12d

Hello everyone!
As you may be aware I have been accepted in this year’s GSoC 5 under the mentorship of @hkvzjal and @fxm.

I will be giving weekly updates in this thread with it’s relevant PR’s, discussions etc

The initial plan is to start with implementing some `path` functions useful for further functionality taking inspiration from stdlib_os 8 ’s `os_path` module developed by @MarDie, @Arjen and @awvwgk and from how other languages like Python, Go, Rust etc handle it.

If you have any feedback, suggestions, questions etc do let me know!
Thank you!

PRs comming up

`stdlib_system`: essential path functionality #999

Open wassup05 wants to merge 5 commits into `fortran-lang:master` from `wassup05:path`

Conversation 0 Commits 5 Checks 16 Files changed 10

wassup05 commented 4 hours ago Contributor

A few path related functions for ease of future functionality have been added.

- `joinpath` : joins the given paths according to the platform's `path-separator` .
- `operator(/)` : as was suggested in the Fortran discourse here an operator is also provided for the same functionality
- `splitpath` : splits the path following the last `path-separator` and returns the `head` and `tail` .
- `basename` : just returns the `tail` of `splitpath`
- `dirname` : just returns the `head` of `splitpath`

The `pathsep` `parameter` contains either `/` or `\` depending on the platform and is a compile-time constant now, so is the `parameter` , `ISWIN`

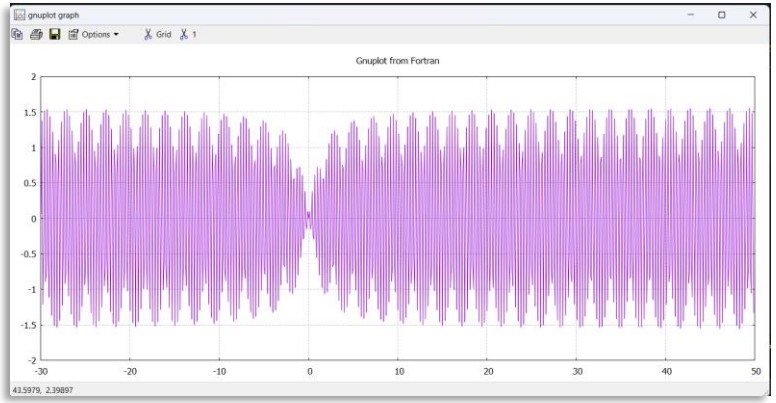
Stdlib system

Plotting with pipes thanks to stdlib process management … an idea for a Fortran plotting framework ?

Inspiration: https://cyber.dabamos.de/programming/modernfortran/gnuplot.html#process

```fortran
program main
  implicit none
  integer,          parameter :: N       = 800
  real,             parameter :: XSTEP   = 0.1

  integer :: i
  real    :: x(N), y(N)

  ! Generate plotting data.
  do i = 1, N
     x(i) = -30 + ((i - 1) * XSTEP)
     y(i) = sin(x(i) * 20) * atan(x(i))
  end do

  ! Plot the data as line chart.
  call plot(x, y, 'Gnuplot from Fortran')
contains
```

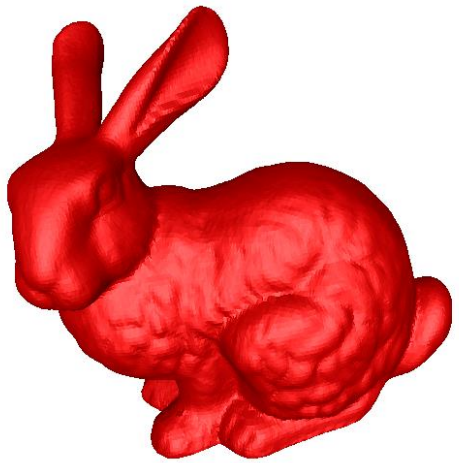
```fortran
subroutine plot(x, y, title)
    use stdlib_system
    use stdlib_strings, only: to_string
    use iso_c_binding
    !! Opens pipe to Gnuplot, writes Gnuplots settings, and plots
    type(process_type) :: p
    real,             intent(inout) :: x(:)  !! X values.
    real,             intent(inout) :: y(:)  !! Y values.
    character(len=*), intent(in)    :: title !! Plot and window title.

    character(len=80) :: buffer
    integer           :: i, rc
    character(len=80) :: args
    character(len=:), allocatable :: stdin

    args = 'gnuplot -persist '

    !> settings
    stdin = 'reset session'//c_new_line
    !stdin = stdin//'set terminal qt'//c_new_line
    stdin = stdin//'set title "' // trim(title) // '"'//c_new_line
    stdin = stdin//'set grid'//c_new_line
    stdin = stdin//'set nokey'//c_new_line//c_new_line
    ! Output X, Y data.
    stdin = stdin//'plot "-" using 1:2 with lines'//c_new_line
    do i = 1, size(x)
       stdin = stdin//to_string(x(i), '(f12.8)')//' '//to_string(y(i), '(f12.8)')//c_new_line
    end do
    stdin = stdin//'e'//c_new_line

    ! End plot data.
    p = run(args, stdin=stdin, want_stdout=.true.)
    if (is_completed(p)) then
       print *, "Process completed successfully. The current directory: "
       print *, p%stdout
    else
       print *, "Process is still running (unexpected)."
    end if

end subroutine plot

end program main
```



Mini app demo

TRANSVALOR

Mini app



Challenges:

* Read an ASCII file containing the mesh data
* Use the high-level linalg API for the eigenvalues decomposition
* Get a sparse matrix representation of the bunny

1. Create project

```
fpm new mini_app
```

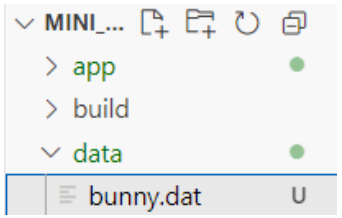
2. Modify the fpm.toml file to use stdlib as meta-package (or local relative path)

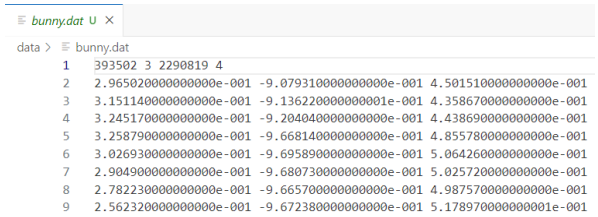
```
[dependencies]
stdlib = "*"
```

OR

```
[dependencies]
stdlib = { path = "../stdlib" }
```

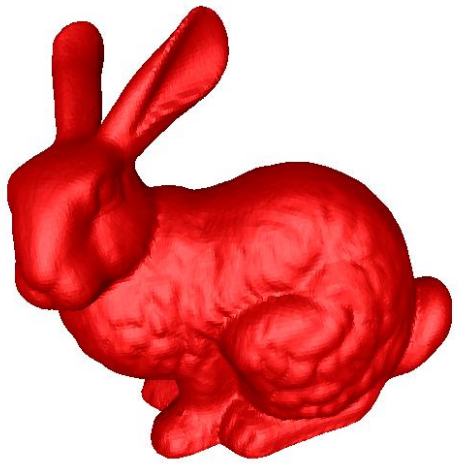
3. Bring custom data into the project





Custom mesh with
393502 nodes
2290819 tetrahedra

Mini app



Two reading methods:

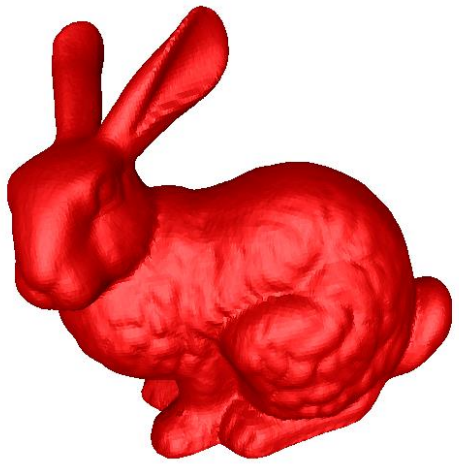
Using Fortran intrinsic read

```
!---------------------------------------- Read dimensions
read(u,*) num_nodes, dim, num_elems, n_per_elt

if( n_per_elt == 0 .OR. dim == 0 ) return

allocate( coordinates(dim, num_nodes) )
allocate( connectivity(n_per_elt,num_elems) )
!---------------------------------------- Read Nodes
do i = 1, num_nodes
  read(u,*) coordinates(1:dim, i)
end do
!---------------------------------------- Read Elements
do i = 1, num_elems
  read(u,*) connectivity(1:n_per_elt, i)
end do
```

Using stdlib's to_num_from_stream

```
!---------------------------------------- Load file in a single string
inquire(unit=u, size=fsze)
allocate(character(fsze) :: ff)
read(u) ff
close(u)
!---------------------------------------- Read dimensions
ffp => ff(1:)
num_nodes = to_num_from_stream( ffp , num_nodes )
dim       = to_num_from_stream( ffp , dim )
num_elems = to_num_from_stream( ffp , num_elems )
n_per_elt = to_num_from_stream( ffp , n_per_elt )

if( n_per_elt == 0 .OR. dim == 0 ) return
allocate( coordinates(dim, num_nodes) )
allocate( connectivity(n_per_elt,num_elems) )
!---------------------------------------- Read Nodes
do i = 1, num_nodes
  do k = 1, dim
    coordinates(k, i) = to_num_from_stream( ffp , coordinates(k, i))
  end do
end do
!---------------------------------------- Read Elements
do i = 1, num_elems
  do k = 1, n_per_elt
    connectivity(k,i) = to_num_from_stream( ffp , i )
  end do
end do
```

Mini app



Two reading methods:

Using Fortran intrinsic read

```
!------------------------------------------------------------------------
!> Load the mesh
block
  integer :: err
  real(dp) :: time_start, time_finish
  call cpu_time(time_start)
  err = load_mesh( './data/bunny.dat' , coordinates , connectivity)
  call cpu_time(time_finish)
  write(*,*) 'File loading time [s]:', time_finish - time_start
end block
```

File loading time [s]: 1.75000000000000

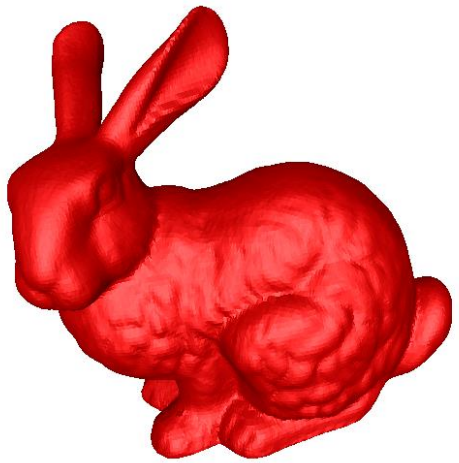
Using stdlib's to_num_from_stream

```
!------------------------------------------------------------------------
!> Load the mesh
block
  integer :: err
  real(dp) :: time_start, time_finish
  call cpu_time(time_start)
  err = load_mesh_stream( './data/bunny.dat' , coordinates , connectivity)
  call cpu_time(time_finish)
  write(*,*) 'File loading time [s]:', time_finish - time_start
end block
```

File loading time [s]: 0.234375000000000

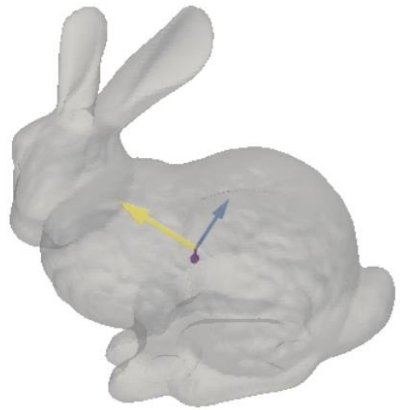
x7 faster

Mini app



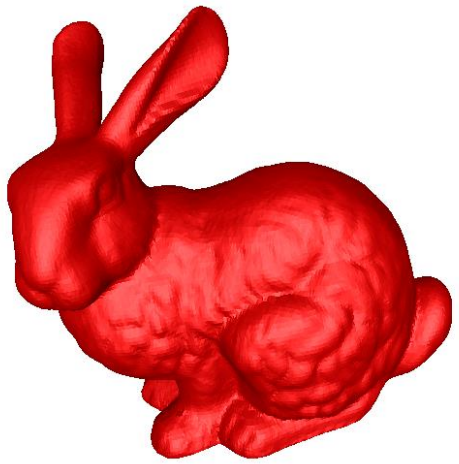
Covariance and Ellipsoidal envelope

```
! Compute centroid
N = size( coordinates, dim = 2 )
centroid(1:3) = stdlib_sum( coordinates , dim=2 )
centroid(1:3) = centroid(1:3) / N

! Compute covariance
covariance = 0._dp
do concurrent( node = 1:N, j = 1:3, i = 1:3 )
  covariance(i,j) = covariance(i,j) + (coordinates(i,node)-centroid(i)) &
                                    * (coordinates(j,node)-centroid(j))
end do
covariance = covariance / N

! Eigen values and Eigen vectors
call eigh(covariance, S, U)
S = sqrt(S)
```



```
Bunny centroid :
    -8.7728E-02     -2.7064E-01      1.5840E-01

1st axis:
     2.7939E-01      8.8100E-02      1.5809E-01      9.8349E-01
2nd axis:
     3.4811E-01      6.2281E-01      7.6179E-01     -1.7825E-01
3rd axis:
     5.0802E-01     -7.7739E-01      6.2823E-01     -3.1349E-02
```

Mini app



```
subroutine coo_from_cells(COO,cells,num_points,num_cells,shift,symtype,selfloop)
  use stdlib_sparse
  class(COO_type), intent(inout) :: COO
  integer, intent(in) :: cells(:,:) !! Cell to point connectivity (a cell can be a volume element, a face, an e
  integer, intent(in) :: num_points !! total number of points grouped by the cells
  integer, intent(in), optional :: num_cells !! Number of cells, if not given then computed from the size of th
  integer, intent(in), optional :: shift    !! Shift the numbering of points
  integer, intent(in), optional :: symtype  !! 0=> full matrix, 1=> triangular inf, 2=> triangular sup
  logical, intent(in), optional :: selfloop !! include diagonal elements
...
  COO%is_sorted = .false.
  call coo2ordered(COO)
end subroutine
```

```
block
  use stdlib_sparse
  type(COO_dp_type) :: COO
  type(CSR_dp_type) :: CSR
  real(dp) :: time_start, time_finish

  call cpu_time(time_start)
  call coo_from_cells(COO, connectivity, num_points = N)
  call cpu_time(time_finish)
  write(*,*) 'Convert cells to COO [s]:', time_finish - time_start

  call cpu_time(time_start)
  call coo2csr(COO,CSR)
  call cpu_time(time_finish)
  write(*,*) 'Convert COO to CSR   [s]:', time_finish - time_start

  write(*,*) ''
  write(*,*) 'non-zeros vs N^2  :',CSR%nnz, CSR%nrows*CSR%ncols
  write(*,*) 'Storage efficiency:',real(CSR%nnz)/real(CSR%nrows*CSR%ncols)
end block
```

```
Convert cells to COO [s]:  0.656250000000000
Convert COO to CSR   [s]:  1.562500000000000E-002

non-zeros vs N^2  :     3068466   225001348
Storage efficiency:  1.3637546E-02
```

Lessons Learned

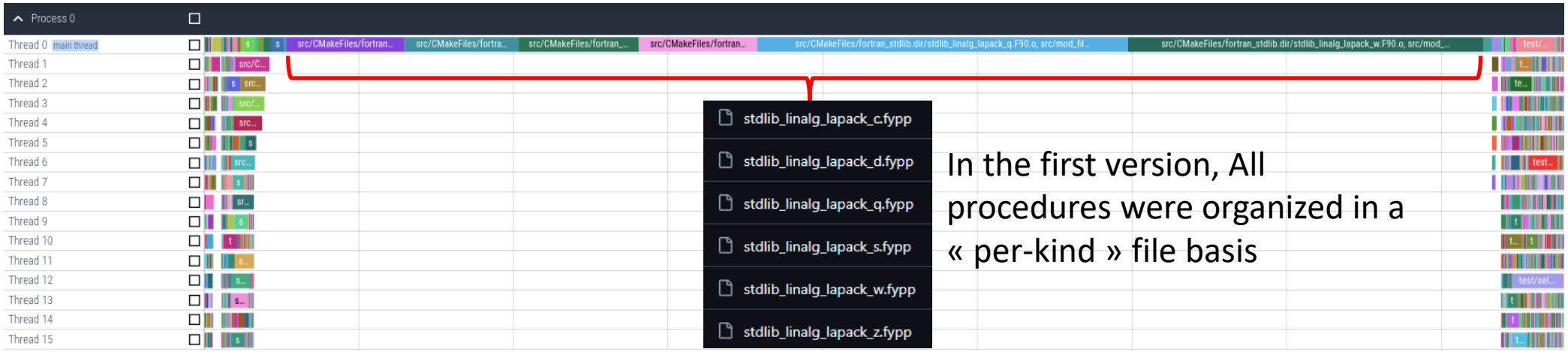
TRANSVALOR

Build efficiency

After first adding BLAS/LAPACK into stdlib we realized that the build was very inneficient …

Parallel build example with CMake + Ninja + gfortran https://ui.perfetto.dev/

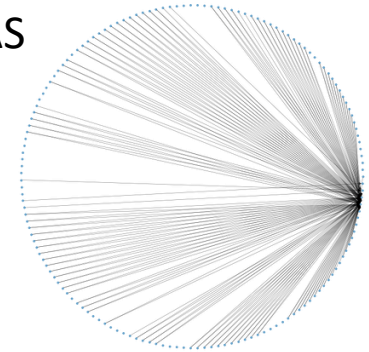
Note: Ninja tracer now default in Cmake 4.0

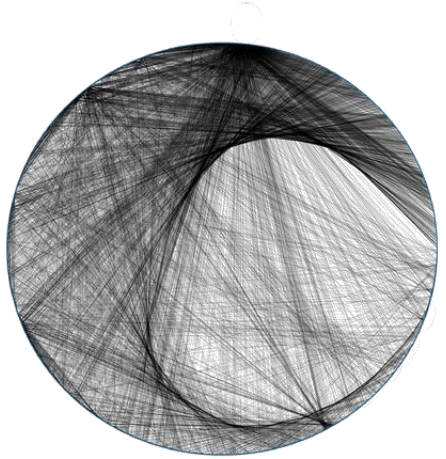


In the first version, All procedures were organized in a « per-kind » file basis

Can a dependency graph help?

BLAS





LAPACK

Some silos can be seen …
But not there yet

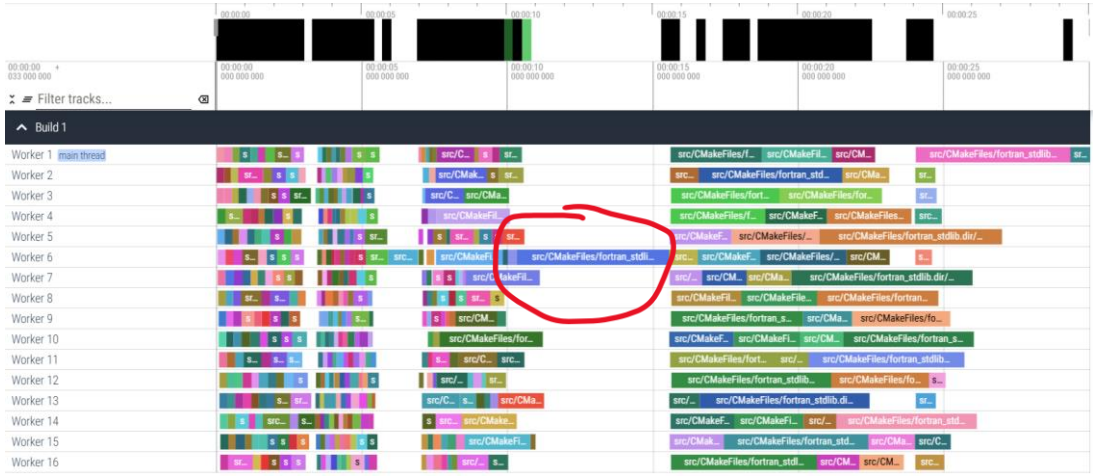
Build efficiency

We iterated over the following re-structuring (together with Ivan Pribec and Federico Perini)

1. Follow LAPACK's grouping

•LAPACK
- Linear solve
- Least squares
- Orthogonal/unitary factors (QR, CS)
- Non-symmetric eigenvalues
- Hermitian/symmetric eigenvalues
- Singular Value Decomposition
- BLAS-like procedures
- Auxiliary routines

•BLAS
- Scalar operators
- Level 1 BLAS: vector ops
- Level 2 BLAS: matrix-vector ops
- Level 3 BLAS: matrix-matrix ops



Iteration with several LAPACK submodules with a **single LAPACK header module**

2. Use modules for procedures interfaces and submodules for the actual implementations

Iteration by further splitting into several LAPACK header modules

Build accelerated x6 with respect to the original version and even faster than the reference LAPACK

Contributing

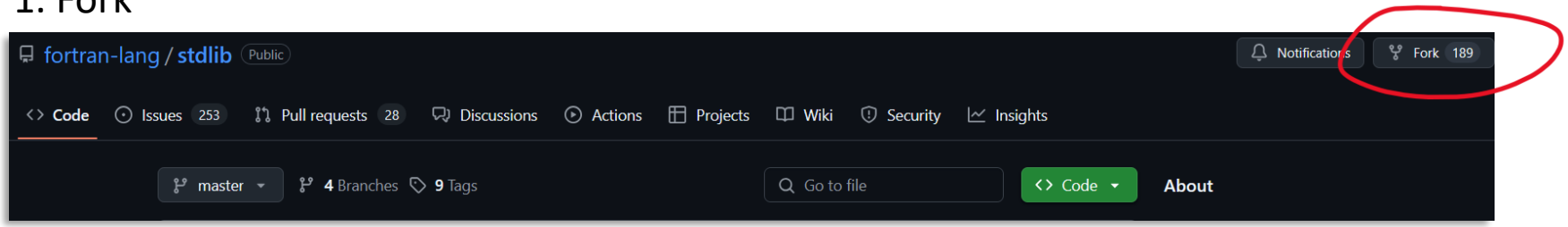
Contributing to stdlib

A short wish list of contributions

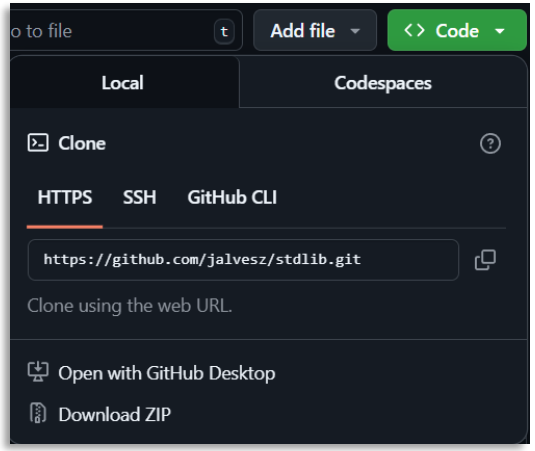
- Linalg: Sparse Direct Solvers
- Linalg: Preconditioning (dense and sparse)
- Linalg: Graph-ordering algorithms: AMD, Cuthill-McKee, Reverse CM, Minimun Degree
- Linalg: Special Matrices (started by Jean-Christophe LOISEAU) https://github.com/fortran-lang/stdlib/pull/957
- Linalg: IO for Matrix Market https://github.com/fortran-lang/stdlib/issues/763
- Linalg: Link to external high-performance libraries for sparse matrices https://github.com/fortran-lang/stdlib/issues/934 https://github.com/fortran-lang/stdlib/pull/844
- CI/CD: add tooling for improved code quality https://github.com/fortran-lang/stdlib/issues/937 https://github.com/fortran-lang/stdlib/issues/944

Contributing to stdlib

1. Fork



2. Clone a local copy from your fork




```
git clone https://github.com/<user>/stdlib.git
cd stdlib
```

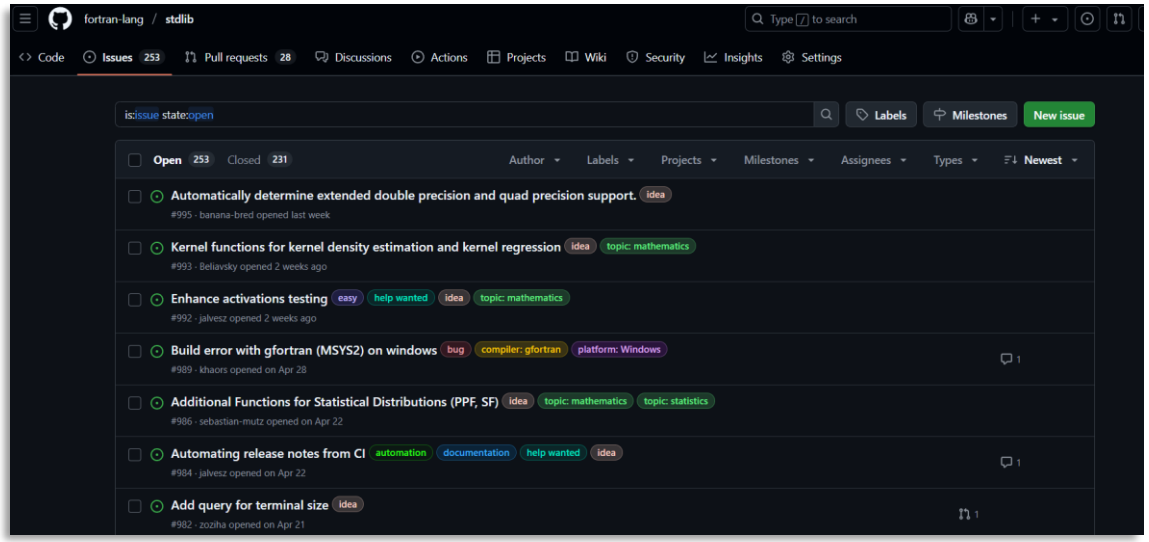
3. Start testing with either CMake or FPM

```
cmake –B build
cmake –build build
```

```
python config/fypp_deployment.py
fpm build --profile release
```

Contributing to stdlib

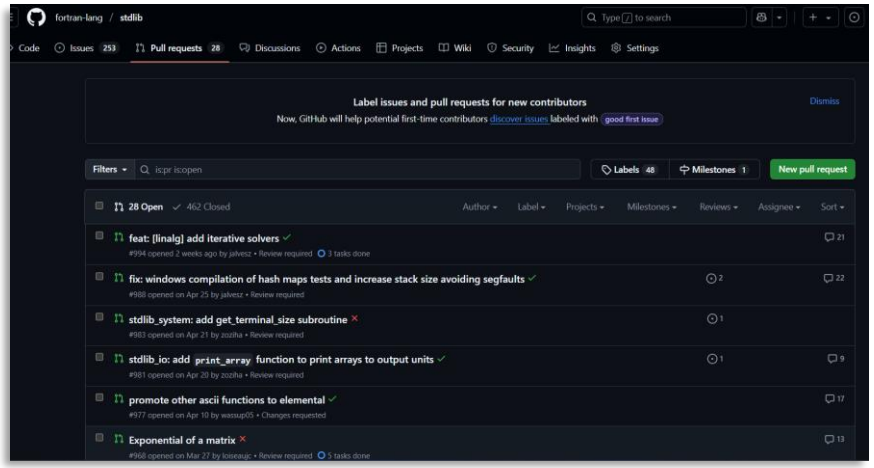
4. Reporting issues / Suggesting new features

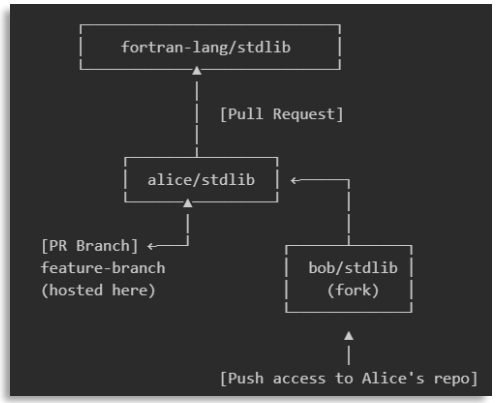


Reminder: currently stdlib is being driven by several ***volunteers*** from the community. You might not get an instantaneous answere to your query.

Suggestion: open a post at the fortran-lang.discourse → it has larger visibility to the community compared to the issue tracker.

5. Collaborating with an onpen PR





A collab script to get you started

```
./.github/collab.sh
```

```
Enter the GitHub username of the fork owner (e.g., alice): alice
Enter the PR branch name (e.g., feature-branch): feature-branch
```

Take-aways

TRANSVALOR

Take-aways

- Stdlib offers a « high-level » experience of coding
- It incorporates modern practices as a reference and playground
- « By » and « For » the community