2-D vortex patch fortran 90 program

Tae Eun Kim

March 7, 2018

Contents

1 Todo's

- ☒ Useful general modules
  - ☒ constants: pi, i, e, line(—...), ...
  - ☒ function(vnorm) 1-norm / 2-norm / ~~infinity-norm~~
  - ☒ subroutines: linspace (a+(b-a)*i/n, i=0,n) and linspace1 (a+(b-a)*i/n, i=0,n-1)
  - ☒ subroutines: timestamp
- ☒ subroutines and functions
  - ☒ reading and writing data
  - ☒ point evaluation
  - ☒ velocity (integration) - new version with efficient alternating point quadrature
  - ☒ residual (as a function)
  - ☒ vsolver - newton continuation
  - ☒ c_0 calculation
  - ☒ transfer matrix M calculation - efficient version using fft
  - ☒ j(nu) calculation
- ☒ newton continuation method
  - ☒ LINPACK
  - ☒ LAPACK
  - ☒ save solution
- ⊟ fft
  - ☒ Using fftw
  - ☒ double Fourier series: two 1-D FFT & one 2-D FFT

    - ☐ FFT based on shifted ponits
    - ☐ Self-contained code? Is it possible?

- ☒ Is it possible to compile files in subdirectory and to save binary files in a subdirectory? - see below
- ☐ Find out why coefficients have discrepancy (~1e-12) between matlab and fortran
- ☐ plotting using gnuplot

2 Directory organization

2.1 Project directory

2.2 TODO More specialized and cleaner structure

See if I can clean it up more using functionalities of `make`.

```
../
 data
 doc
 src
 bin
 obj
 fig

6 directories
```

3 Program organization

3.1 Modules

ISSUE. When tangled, emacs automatically wraps the module file with `program main` and `end program main`, which need to be deleted manually.

3.1.1 Constants

- set real kind (single or double precition)
- pi, i, eps

3.1.2 Basic routines - printing, linspace, etc

- [SR] writing complex data
- [FN] linspace, linspace1, linspaceh
- [FN] identity matrix, zero array constructor
- [FN] vector infinity-norm calculator
- [FN] generating data filename
- [SR] time stamp
- [FN] matrix infinity-norm calculator
- [SR] generating data filename

3.1.3 Notes:

- When a function defined externally outside the main program, it appears that gfortran compiler wants to have an `interface` block. In case a function defined in a module is called, it works fine without `interface` block.

3.2 Main program

3.2.1 solver program

3.2.2 transfer matrix calculation

3.2.3 fft program

3.2.4 calculation of bounds

3.2.5 testing program

3.3 Subroutines and functions

3.3.1 Reading data

3.3.2 Writing data

3.3.3 Point evaluation

3.3.4 Velocity calculation

3.3.5 Residual calculation

3.3.6 Vortex patch solver (Newton continuation)

3.3.7 FFT and T-matrix calculation

4 Data file structure

4.1 Naming convention

`vp_bI_nEE_rDDDD.dat` where

- `bI` describes how β value was obtained: $\mathrm{I} = 0 : \beta = 0$ $\mathrm{I} = 1 : \beta = (1 - \sqrt{1-\rho^2})/\rho$ $\mathrm{I} = 2 : \beta = (1 - 2\sqrt{1-\rho^2})/\rho$
- `nEE` indicates that $n = 2^{\mathrm{EE}}$.
- `rDDDD` represents the value of $\rho = 0.\mathrm{DDDD}$.

4.2 Example

An example data file `vp_b0_n07_r5000.dat` may look like

line	file	note
1	128	n
2	0.5000	rho
3	0.0000	beta
4	2.7814117251577763 (-01)	U
5	2.3675575948962696 (-01)	a_1
6	-6.6992137885540828 (-02)	a_2
⋮	⋮	⋮
131	2.8142318944085296 (-20)	a_{n-1}

Our `data` directory looks like this:
Here is one of the actual data file:

5 LINPACK

5.1 Useful subroutines: double precision

- DGECO: calculates condition number
- DGEDI: calculates determinant
- DGESL: solves A*X = B

5.2 Example

5.3 Notes

- When using Burkardt's `linpack_d.f90`, make sure to link `lapack` as it is not self-contained.

  ```
  gfortran -o main.exe main.f90 linpack_d.f90 -framework Accelerate
  ```

- However, the quadruple precision library `linpack_q.f90` is self-contained:

  ```
  gfortran -o main.exe main.f90 linpack_q.f90
  ```

- When using `linpack`, compile with either
  - linpack.f (Fortran77)
  - $\text{linpack}_{\text{d.f90}}$ (Fortran90) with `-framework Accelerate` flag.

  Even if the main program follows Fortran90 standards, `linpack.f` works seamlessly.

- **Update** The source files `linpack*` are simply collection of routines (dependencies) required for `DGECO`, `DGEDI`, and `DGESL`. Some of them are again dependent on some routines of `blas` library. The required routines are identified and combined into a single source file for both `linpack` and `lapack`.

6 LAPACK (modern)

6.1 Using LAPACK in Mac

Mac supplies a copy of LAPACK compiled and optimized for Apple hardwares and it is easily available as a library. In order to link/load the library, include `-framework Accelerate` compilation flag, e.g.,

The library is located in the system directory /System/Library/Framework.

6.2 Useful subroutines and their usage

Let $A \in \mathbb{R}^{n \times n}$.

- DGETRF: LU-factorization of A
- DGECON: calculates condition number of A
- DGETRI: calculates the inverse A^{-1} using LU-decomposition
- DGETRS: solves $Ax = b$ via Gaussian elimination, i.e. LU-factorization
- **Note.** One of the inputs, `anorm`, for `dgecon` must be calculated before calling the routine. A function calculating matrix infinity-norm has been included in my module, currently named as `mnorm`.

6.3 Example snippet

7 FFTW

7.1 Installation and basic usage

On Mac, I used `homebrew`

The files `libfftw3xxx.a` are saved in /usr/local/lib directory. At link time, use `-l` flag as follows:

The program file `myprog.f90` should contain a line which declares variables used.

On my linux machine running ArchLinux, I installed it using `packer`:

7.2 Variable declaration

The file `fftw3.f90` declares variables needed for execution of `fftw` routines:

7.3 Forward and backward 1-D (complex) DFT routines

The **forward DFT** of 1-D complex array X of size n calculates an array Y of the same dimension where

$$Y_k = \sum_{j=0}^{n-1} X_j e^{-2\pi ijk/n} \,.$$

The **backward DFT** computes

$$Y_k = \sum_{j=0}^{n-1} X_j e^{2\pi ijk/n} \,.$$

Note that `fftw` computes unnormalized transforms. So Fourier series coefficients can be approximated using the forward DFT with $1/n$. The inverse discrete Fourier transform is numerically calculated with the backward DFT without any normalization.

Note also that an output of the forward DFT are ordered so that the first half of the output corresponds to the positive modes while the second half to the negative ones in backwards order; this is due to the n-periodicity of Y_k in its index.

Example. When $n = 8$, we have the following correspondence between indices (k), mode numbers, and Fortran indices:

k	mode	fortran
0	0	1
1	1	2
2	2	3
3	3	4
4	Nyquist	5
5	-3	6
6	-2	7
7	-1	8

In general:

k	mode	fortran
0	0	1
1	1	2
:	:	:
n/2-1	n/2-1	n/2
n/2	Nyquist	n/2+1
n/2+1	-n/2+1	n/2+2
:	:	:
n-1	-1	n

Snippet.

7.4 Forward and backward 2-D (complex) DFT routines

The **forward DFT** of $n \times n$ 2-D complex array X calculates an array $Y \in \mathbb{C}^{n\times n}$ where

$$Y_{j,k} = \sum_{m=0}^{n-1}\sum_{l=0}^{n-1} X_{l,m} e^{-2\pi i(jl+km)/n} \,.$$

The **backward DFT** computes

$$Y_{j,k} = \sum_{m=0}^{n-1}\sum_{l=0}^{n-1} X_{l,m} e^{2\pi i(jl+km)/n} \,.$$

Note that these are simply the separable product of 1-D transforms along each dimension of the array X, that is, along the columns and rows of X.

Once again, `fftw` computes unnormalized transforms and so double Fourier series coefficients can be approximated using the forward DFT with $1/n^2$. The inverse discrete Fourier transform is numerically calculated with the backward DFT without any normalization.

Snippet

7.5 Shifted FFT

Using the approximating nature of DFT on physical data against the coefficients of Fourier expansion, we may utilize FFT routines on data obtained on half-step shifted grids on $[0, 2\pi)$.

For the sake of illustration, consider the 1-D FFT situation where $X^{(s)} \in \mathbb{C}^N$ is a vector of point values at $2\pi(j + 1/2)/N$ for $0 \le j < N$ and $Y^{(s)}$ is the result of the forward FFT on $X^{(s)}$, i.e.,

$$Y_j^{(s)} = \frac{1}{N}\sum_{k=0}^{N-1} X_k^{(s)} e^{-2\pi ijk/N} \,.$$

By adjusting the phase of complex exponentials, we can interpret them in term of approximate Fourier coefficients of the underlying function for the $X^{(s)}$ data. Keeping in mind the aliasing errors associated with the discrete Fourier transforms, i.e., the N-periodicity over the index j, we observe that

- For $0 \leq j < N/2$,
$$Y_j^{(s)} \approx Y_j e^{-ij/N} \, ,$$
- For $N/2 < j < N$,
$$Y_j^{(s)} \approx Y_j e^{-i(j-N)/N} \, ,$$

The second case was considered with

8 Compiling with `gfortran`

9 Makefile

9.1 Editing a make file in Emacs/Org-mode

- When a `make` source code written in org-mode src block is tangled, tabs are converted to spaces. One can manually `M-x tabify` the entire file.
- A makefile whose file name is not `Makefile` will not be in `makefile-mode` automatically. Set the mode by `M-x makefile-mode`.
- In order to run a makefile with filename other than `Makefile`, use `make -f filename`.

9.2 Some idea from StackOverflow

From Specify directory where gfortran should look for modules

```
You can tell gfortran where your module files (.mod files) are located with the -I comp

I use these to place both my object (.o files) and my module files in the same directo

SRC = /path/to/project/src
OBJ = /path/to/project/obj
BIN = /path/to/project/bin
```

```
gfortran -J$(OBJ) -c $(SRC)/bar.f90 -o $(OBJ)/bar.o
gfortran -I$(OBJ) -c $(SRC)/foo.f90 -o $(OBJ)/foo.o
gfortran -o $(BIN)/foo.exe $(OBJ)/foo.o $(OBJ)/bar.o
While the above looks like a lot of effort to type out on the command line, I generally

Just for reference, the equivalent Intel fortran compiler flags are -I and -module. Es
```

10 Fortran 90 programming tips

10.1 Notes on precision in fortran90+

At the beginning of program, set real precision, which is of integer type, to be

- 4 : single
- 8 : double
- 16 : quadruple

For example, declare

Then the precision of a real or complex variable can be declared by: or simply by

In the body of program, the precision of a floating point number can be set by suffixing with the precision parameter, e.g.

- 1.0_4 : single, same as 1.0
- 1.0_8 : double, same as 1.d0
- 1.0_{16} : quadruple

Once the precision is stored in the variable, say `rp`, one can simply write `1.0_rp`.

The precision of outputs of an intrinsic function is determined by that of its input(s). This way, we can avoid using old `d`-variations, e.g., `dcos`, `dsin`, `dabs`, etc.

A complex number of certain precision can be constructed using `cmplx` function with the following syntax:

10.2 Nice tips on using `read` and `write` functions

The following is an example of using `write` function to assimilate the functionality of `MatLab`'s `sprintf`.

10.3 `linspaceh` function:

The function `linspaceh(a, b, h)` constructs a vector of uniformly spaced-out points between `a` and `b` with gap `h`. In case `b-a` is not a (numerical) multiple of `h`, then the gap between `b` and the one before will be smaller than `h`. This function is included in mymod.f90.

10.4 I/O formatting

10.4.1 `print`

Printing out to terminal: `print format_specifier, i/o_list`, e.g. `print *, 'hello world'`

10.4.2 `write`

10.5 Arrays - basics

- In Fortran 90+, one can construct arrays with inline do-loops, a.k.a., implied do-loops. For example,
- Of course, `i` and `j` need to be declared integers and `v` and `w` as integer/real/complex arrays of appropriate dimensions. Note below how types are cast:
- `size` and `shape`

10.6 Arrays - assignment

Many functions on arrays behave similarly to those of `MatLa`.

10.7 Arrays: `transpose` and `spread`

10.8 Arrays: general concatenation using `reshape`

10.9 Arrays: trick to calculate maximal value of an array using `reshape`

10.10 Characters

10.10.1 Example

10.10.2 Notes

- Use `character(len=*)` when the length of a character string is not known.

10.11 Important notes on `interface`

When one intends to input an array of arbitrary size into a routine, the *assumed shape* technique turns out to be quite advantageous. Consider the following sample program:

Note that the `interface` block is required. In the block, only the arguments to the routines need be type-cast.

In order to avoid writing such blocks over and over, utilize a module structure as follows:

11 Testing area