1 Package Distribution

There are a number of required packages to run this project, such as numpy, scipy and fft packages. In order to have everyone on the same packages and package versions, the easiest is to use a *virtual environment*, `virtualenv`. There is more information here on the computer documentation wiki. Below I've given a quick explanation how it would be used

- Make a directory somewhere on your local machine (not in the project folder) to store the virtual environment and cd into it, like this:

  ```
  $ mkdir python_virtual_env && cd python_virtual_env
  ```

  This folder is where the packages etc will end up being installed. If you would like to remove the virtual environment, all you have to do is delete the folder it is in.

- Now we create the virtual environment. We make a virtualenvironment named `holo-env` with the command

  ```
  $ virtualenv --python=python3.6 holo-env
  ```

  You will see that a directory `holo-env` has been created in the folder. By default, the only packages that are installed are those that are required to install other packages.

- To start using the virtualenv, you should run the following:

  ```
  $ source holo-env/bin/activate
  ```

  You will that at it will say `(holo-env)` at the start of your next line in the terminal. This means that you are now in the virtual environment. Python is then simply run with

  ```
  $ python infile.py
  ```

  You can see the packages that are installed with

  ```
  $ pip list
  ```

- Now you can install any package that you like. The packages required to run the project are in the root folder of the project in `requirements.txt`. You can install the packages in there using

  ```
  $ pip install -r requirements.txt
  ```

  If you want to add a package to the requirements, you should install it in your virtual environment and export the requirement using

```
$ pip freeze > requirements.txt
```

That is really all there is to it. To exit the virtual environment, simply type

```
$ deactivate
```

and you will see that your command line returns to normal.

2 Introduction

3 Design of the package

The package consists of separate modules as shown on Diagram 1. It consists of a set of initialization blocks, which include importing the form of equations from Mathematica (or elsewhere), setting up the grid and choosing the initial solution and the parameters of the problem; and the internal loop, where the solution is found after several iterations. For performance purposes it is important to optimize the internal loop, while the initialization blocks only need to preserve the precision of data.

Update 17 April: The current structure is as follows. The problem is defined by a set of equations (EoMs, coefficients, etc), the input parameters (dimensions, grid spacings, etc). Currently, this is being handled by:

- Load in the ini file into a setup class. The base setup class has all relevant information to execute a simple nonlinear solve where we just have a set of dynamical fields and constants. In the holo_setup module you'll see the Setup class, this is the base class. The easiest way to handle fermions on top of a fixed background is by extending the base class. This is also done in the holo_setup file, and the only things it needs extra are the ability to read background data and a modification in terms of the variables that are used.

- The main solver routine is now in a class, NonlinearSolver, which has the main method "solve". This is the main routine: it evaluates derivatives, makes the updates, etc. Preconditioning using a lower-order approximation of the operator has also been implemented. convergence, there are callbacks available to do additional processing (e.g. smoothing, checkpointing, enforcing constraints) to aid in the evaluation process. There is also the convergence hook, which naturally checks for convergence etc. I have still to implement a working version of it, but it should not take much time. I have included an example of a 'timed hook', which could for example be used to checkpoint the run every so often.

- Fermions. Fermions I have put in a different section of code because it is so different. It requires its own setup, it requires its own file-reading utilities etc. It is working properly now though, and we should be able to use it.

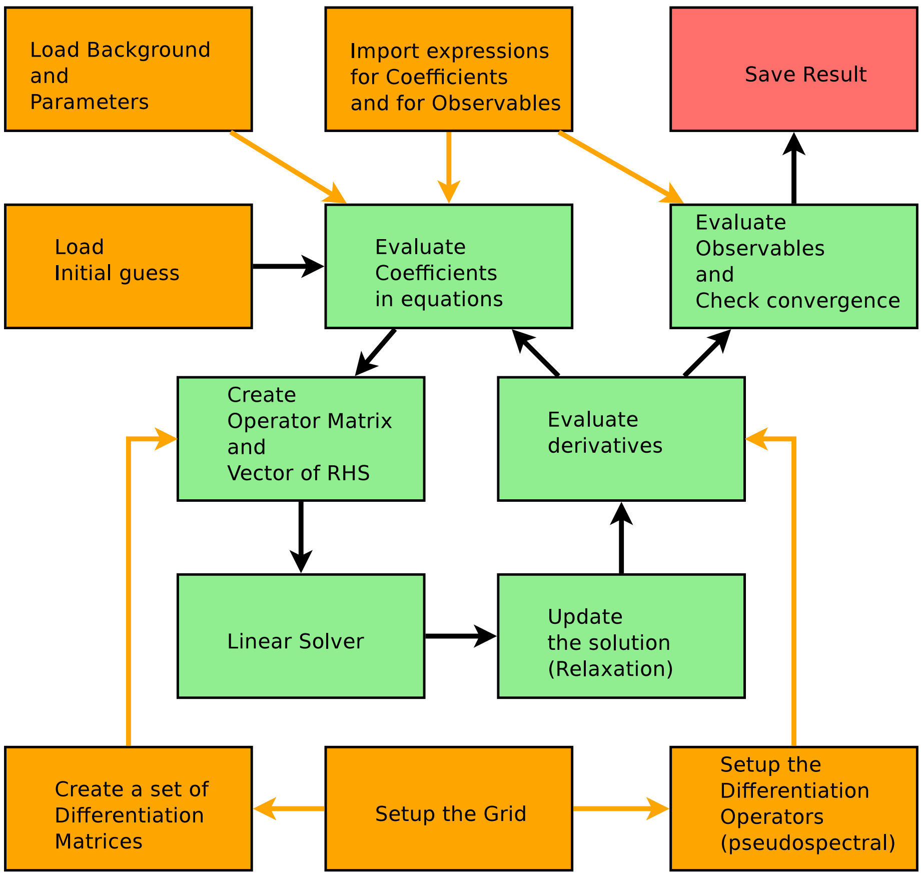
Load Background and Parameters
Import expressions for Coefficients and for Observables
Save Result
Load Initial guess
Evaluate Coefficients in equations
Evaluate Observables and Check convergence
Create Operator Matrix and Vector of RHS
Evaluate derivatives
Linear Solver
Update the solution (Relaxation)
Create a set of Differentiation Matrices
Setup the Grid
Setup the Differentiation Operators (pseudospectral)

Figure 1. **Principal structure of the package.** Yellow blocks – initialization, Green blocks – parts of the loop, which deserve optimization. Arrows show the flow of data. The data structures are explained in the text.

4 Initialization blocks

4.1 Input/Output of data

This is relevant to the blocks **Load Background and Parameters**, **Load Initial Seed** and **Save Result**. The data should be organized in a unified way as discussed in 6.1

4.2 Import expressions for Coefficients and for Observables

This is a block which converts data from Mathematica to Python. It works together with the Mathematica counterpart, which prepares the arrays of coefficients in the appropriate way.

4.3 Setup the Grid

This block should create the appropriate grids, taking as input the number of points in each direction, type of the grid (either **Homogeneous** or **Pseudospectral**) and type of the boundary conditions (either **periodic** or not).

4.4 Create a set of Differentiation matrices

It takes the gridpoints and the creates the "next neighbour" order diff matrices for a set of derivatives. The outcome is organized as an array with matrices, which include in various dimensions 3, 6 or 10 of them

$$\text{1D}: \quad \{\mathbf{1}, \mathbb{D}_x, \mathbb{D}_{xx}\} \tag{4.1}$$

$$\text{2D}: \quad \{\mathbf{1}, \mathbb{D}_x, \mathbb{D}_y, \mathbb{D}_{xx}, \mathbb{D}_{yy}, \mathbb{D}_{xy}\} \tag{4.2}$$

$$\text{3D}: \quad \{\mathbf{1}, \mathbb{D}_x, \mathbb{D}_y, \mathbb{D}_z, \mathbb{D}_{xx}, \mathbb{D}_{yy}, \mathbb{D}_{zz}, \mathbb{D}_{xy}, \mathbb{D}_{xz}, \mathbb{D}_{yz}\} \tag{4.3}$$

The subtlety here is that the grid may not be homogeneous. In practice we need matrices only for homogeneous and Chebyshev lattices, with either periodic or not boundary conditions.

4.5 Setup the Differentiation Operators

This block prepares the operator which will be used in order to evaluate derivatives with high precision. This is either a set of Fourier transforms for pseudospectral case or high order differentiation matrices for FDD case. It return the set of operators in the same order as (4.1)

5 Loop blocks

5.1 Evaluate Coefficients in equations

This get the expressions for the coordinate dependent coefficients and the values of function and its derivatives on the grid. The latter is organized as a set of d-dimensional arrays, which represent the function itself, and its derivatives in order of (4.1) on the d-dimensional grid. For instance in 2D this is

$$\mathbb{F} = \{F_{ij}, (\partial_x F)_{ij}, (\partial_y F)_{ij}, (\partial_{xx} F)_{ij}, (\partial_{yy} F)_{ij}, (\partial_{xy} F)_{ij}\} \tag{5.1}$$

This block makes as many evaluations as there are grid points at every step. It might deserve optimization through compilation of the coefficient functions, maybe even in C.

It returns the coefficients as arrays of the shape $\{ne, nf * nd, nx, ny, \ldots,$ where ne is the number of equations, $nf * nd$ – the number of different terms in the equation, i.e. nd possible derivatives of nf possible functions.

5.2 Create Operator Matrix and Vector of RHS

This one multiplies the evaluated coefficients by the appropriate differentiation matrices and sums over, creating an BVP operator. It also takes care of implementing the boundary conditions in the right way.

It will also create the vector of right hand sides, which just consists of appropriate nonhomogeneous coefficients evaluated in the previous block.

This one requires clever working with sparse matrices

5.3 Linear Solver

As simple as it is.

5.4 Update the solution

It updates the solution according to the specified rule. For instance for relaxation

$$F_{i+1} = F_i - \frac{3}{7} df \tag{5.2}$$

5.5 Evaluate derivatives

Evaluated the derivatives using the Differential operators defined at Initialization step. Returns the data structure of (5.1)

5.6 Evaluate observables and Check convergence

Checks convergence according to the specified rule and stops the cycle.

6 Data structures

6.1 Data file names ans structure

The solutions are saved in the data files. The file contains the array:

$$\{\text{params}, \text{observables}, \text{solution}\} \tag{6.1}$$

Keeping the information about physical parameters (6.5), observables (6.6) and the solution itself together with coordinate arrays in the form of 6.2.

6.2 Solution on the grid

The solution on the grid is represented as $\{ne + d, nx, ny, \dots\}$ array, where ne is a number of functions to be solved for, $nx1, \dots$ the sizes of grid in corresponding directions and d is the number of dimensions in the problem, so that the last nd entries on the first level are the values of corresponding coordinates of the grid points

6.3 Adjustments of Solution

The adjustments have similar structure to 6.2, but contain only the values for ne functions (without grid coordinates)

$$df \sim \text{solution}[[1;;ne, All, All]] \tag{6.2}$$

Input	**Block**	**Output**
Data file name 6.1	**Load Background** 4.1	Solution on the grid 6.2 Parameters 6.5
Data file name 6.1	**Load Initial Seed** 4.1	Solution on the grid 6.2
Solution on the grid 6.2 Parameters 6.5 Observables 6.6	**Save Result** 4.1	Data file name 6.1
Files with coefficients 6.7	**Import Coefficients** 4.2	Evaluator of coefficients 6.11
Files with observables 6.10	**Import Observables** 4.2	Evaluator of observables 6.11
Grid parameters 6.12	**Setup the grid** 4.3	Coordindate arrays 6.13
Grid parameters 6.12 Coordindate arrays 6.13	**Create Diff.Matrices** 4.4	Set of Diff. Matrices 6.14
Grid parameters 6.12	**Create Diff.Operators** 4.5	Set of Diff. Operators 6.16
Evaluator of coefficients 6.11 Parameters 6.5 Grid parameters 6.12 Coordindate arrays 6.13 Solution with derivatives 6.4	**Evaluate coefficients** 5.1	Set of coefficients 6.8
Set of coefficients 6.8 Set of Diff. Matrices 6.14	**Create operator and RHS** 5.2	Operator and RHS 6.15
Operator and RHS 6.15	**Linear Solver** 5.3	Adjustmets for solution 6.3
Adjustmets for solution 6.3	**Update solution** 5.4	Solution on the grid 6.2
Solution on the grid 6.2 Set of Diff. Operators 6.16	**Evaluate derivatives** 5.5	Solution with derivatives 6.4
Evaluator of observables 6.11 Solution with derivatives 6.4 Adjustmets for solution 6.3	**Evaluate observables** 5.6	Observables 6.6 Stop signal

Table 1. Input and output data structures of the blocks

6.4 Solution with derivatives on the grid

This structure holds the values of the functions together with their derivatives. It is compatible with the symbols used in the exported coefficient arrays (**??**). It is the array of the shape:

$$\mathbb{F} \sim \{nf, nD, nx, ny, \dots\}, \tag{6.3}$$

Where nD is the order of derivative according to (4.1), $nf-$ the number of function in the ansatz and $nx, ny, \dots$ – coordinates on the grid.

6.5 Parameters

These are the physical parameters of the problem, which enter symbolically the equations of motion. For instance temperature, period of the lattice etc.

For a particular case of striped background the set is:

$$\text{params} = \{T, A, k, N_k, N_p, \phi, c_1\}, \tag{6.4}$$

where T is temperature in units of chemical potential μ, A – the amplitude of the lattice, k – the wavevector of the lattice in units of μ, N_k – number of lattice cells in the computational domain, N_p – number of periods of spontaneous structure in the domain, ϕ – the phase in the background lattice and c_1 – the coefficient in CS term.

6.6 Set of observables

These may include free energy, charge density ets, evaluated for particular solution. In the same set one can include the control structures, like accuracy or precision of the final solution, validity of constraints or the number of iterations it took to converge.

6.7 Files with coefficients exported from Mathematica

The coefficients of equations are exported from Mathematica in the file which contains a number of strings. There are $ne * nD * nf$ strings in the coefficient file. These correspond to the flattened array of coefficients of the shape:

$$\text{exported coefficients} \sim \{ne, nD, nf\} \tag{6.5}$$

Where ne is the number of the equation, nf – the number of functions in the ansatz, nD – the order of derivative. More details about these coefficients (called $\mathbf{X}$) are in the lecture notes [1]

The lines include the recognizable mathematical functions in FORTRAN style, the symbols for physical parameters 6.4 and the symbols for background functions and their derivatives. The last one looks like

$$\texttt{f}[nf, nD] \tag{6.6}$$

Where the first index is the number of function in ansatz and the second is the order of derivative according to (4.1).

There are several files with coefficients, representing the sytem of equations to be solved internally (`CoefEOMsI`), and on the boundaries.

Similar is the structure of the exported expressions for nonlinear Right Hand Sides, except that in these files there are only ne lines, containing the nonlinear equations in terms of `gfi`.

6.8 Set of evaluated coefficients

The coefficients are evaluated on the grid and are first arranged in the array:

$$\texttt{coef} \sim \{ne, nD, nf, nx, ny, \dots\} \tag{6.7}$$

This shape corresoponds to the exported coefficients (6.5), evaluated on the grid points.

6.9 Output of the background solver

Eventually, all IO will be done using the HDF5 file format (see e.g. h5py documentation). In this format, we can directly save and load numpy arrays. Furthermore, it can handle multidimensional arrays very well, and it has a very logical structure. You save data in datasets, which can have other data as attributes. We will save the following parameters:

Data	Type	Location	Attributes
Field Solutions	numpy ndarray	\field	periodicity, gridTypes, nDims, nFields, nEOM
Deriv Methods for EOM	list of string	\controls\EOMDerivMethods	None
constants	dictionary	\controls\parameters	None
Final update	numpy ndarray	\controls\	None
Observables	Dict of name-ndarray	\observables\...	None
Residual of EOM	numpy ndarray	\EOMResidual	None

6.10 Files with observables exported from Mathematica

These files contain a few lines, each of them corresponds to a certain observable expressed in terms of parameters (6.4) and background functions (6.6).

6.11 Evaluator of coefficients or observables

This must be an object, which contains the symbolic expressions for coefficient and is capabel to quickly evaluate the coefficient on the grid in the form (6.7).

6.12 Parameters of the grid

Three arrays, storing the number of points in every direction, the type of grid ("homogeneous" or "pseudospectral") and the type of boundary conditions (periodic: "True" or "False"), it might be a good idea to store explicitely the number of dimentions d and the number of eqiations ne

$$\text{Grid data} \sim \{d, ne, \{nx, ny, \dots\}, \{(\texttt{True}|\texttt{False}), \dots\}, \{(\texttt{True}|\texttt{False}), \dots\}\} \tag{6.8}$$

6.13 Coordinate arrays

This is the array with coordinates in every direction

$$\text{Coord. arrays} \sim \{\{x_1, x_2, \dots\}, \{y_1, y_2, \dots\}, \dots\} \tag{6.9}$$

6.14 Set of Differentiation Matrices

The array containing differentiation matrices (sparse) in the order defined in (4.1).

6.15 Boundary Value Problem operator and vector of RHS

BVP operator is a sparse matrix of the size $\{(ne * nx * ny * \dots) \times (ne * nx * ny * \dots)\}$. Vector of RHS is a dense vector of the compatible size.

6.16 Set of Differentiation Operators

The array containing differentiation operatore in the order defined in (4.1).

References

[1] A. Krikun, arXiv:1801.01483 [hep-th].