A USER GUIDE FOR `OLIM2D` PACKAGE

MANYUAN TAO AND MARIA CAMERON

Contents

1. List of programs 1
2. Compiling and running 2
3. Available options 3
4. Output and plotting graphs in MATLAB 4
5. Changing mesh size, the parameters, and the vector field 5
References 6

In this document, we provide a user guide for `OLIM2D` package implementing the Ordered Line Integral Method for computing the quasi-potential in 2D on a regular rectangular mesh. This package differs from `OLIM` package [3] in that it is programmed using the upgraded hierarchical update strategy described in [1] and data structure similar in `OLIM3D` package [4], which reduces the number of unnecessary triangle updates. Local factoring is not used. Shooting the minimum action path (MAP) is not performed. Matlab programs for plotting graphs are also provided.

1. List of programs

The package contains two C source files implementing the Ordered Line Integral Method with the midpoint quadrature rule for computing the quasi-potential in 2D and two Matlab source files for plotting graphs displayed in this repository:

- `olim2D.c`, a C source code implementing the 2D quasi-potential solver with the upgraded hierarchical update strategy;
- `olim2D_meas.c`, the code `olim2D.c` facilitated for measurements;
- `plot_testK.m`, a Matlab source code for plotting the errors versus the update factor K for mesh sizes $N^2 = (2^p + 1)^2$, $p = 7, 8, 9, 10, 11, 12$, for three examples;
- `plot_err_Kopt.m`, a Matlab source code for plotting CPU time versus the mesh size N, errors versus N, and efficiency, with the optimal choice of update factor K.

The SDE for the evolving system is of the form $d\mathbf{x} = \mathbf{b}(\mathbf{x})\,dt + \sqrt{\epsilon}\,d\mathbf{w}$, $\mathbf{x} \in \Omega \subset \mathbb{R}^2$, where $\mathbf{b}(\mathbf{x})$ is a continuously differentiable vector field, $\mathbf{w}$ is the standard Brownian motion, and $\epsilon > 0$ is a small parameter. The codes `olim2D.c` and `olim2D_meas.c` are designed for computing the quasi-potential with respect to an asymptotically stable equilibrium $\mathbf{x}^*$ of the corresponding deterministic system $\dot{\mathbf{x}} = \mathbf{b}(\mathbf{x})$. The computational domain must be set up so that the equilibrium point $\mathbf{x}^*$ is a mesh point.

2. Compiling and running

We run the codes using the `gcc` C compiler available in the Command Line Tool. Any other C complier should be applicable for running `olim2D.c` and `olim2D_meas.c`. Open Terminal, change the directory to the one where you store files from the package, and type:

```
gcc olim2D.c -lm -O3
./a.out
```

If the program runs normally with the default settings, you should see:

```
chfield = d, fac = 10
x_ipoint: Iindex = 8392704, -1.0000e+00, 0.0000e+00
in param()
in ipoint()
in olim()
The boundary is reached:
7890750 Accepted points, (2048,4095) is accepted, g = 9.9900e-01
NX = 4097, NY = 4097, K = 26
cputime of olim() = 146.744
# of Accepted points = 7890759, umax = 9.9900e-01
Per mesh point:
<#1ptupdates> = 131.21, <#2ptupdates> = 2.42, <#2call> = 4.96
ErrMax = 5.2572e-05, ERMS = 1.8938e-05
Normalized ErrMax = 5.2624e-05, Normalized ERMS = 3.0729e-05
```

Here, `umax` is the maximal value of the computed quasi-potential among `Accepted` and `Accepted Front` points. `NX` and `NY` denote the mesh size of computational domain; `K` is the update factor; `ErrMax` is the maximal absolute error; `ERMS` is the root mean square error.

3. Available options

The codes `olim2D.c` and `olim2D_meas.c` come with several options for choosing the vector field $\mathbf{b}(\mathbf{x})$ and the ratio ρ of the magnitudes of the rotational and potential components of the vector field. The settings in `olim2D.c` and `olim2D_meas.c` are all the same except that `olim2D_meas.c` is designed for testing three specific examples with different mesh sizes and update factors. From now on, we will refer only to `olim2D.c`, and all the line numbers below will be given for `olim2D.c`.

The choice of the vector field is specified by the global variable `char chfield` whose value is assigned at its declaration in line 108. Choose its value out of {`'l'`,`'d'`,`'y'`}. The corresponding vector fields are defined in the function `struct myvector myfield(struct myvector x)` starting on line 136.

- `char chfield = 'l'`, a linear vector field $\mathbf{b}(\mathbf{x}) = \begin{bmatrix} -2x_1 - \rho x_2 \\ 2\rho x_1 - x_2 \end{bmatrix}$.
- `char chfield = 'd'`, a nonlinear vector field $\mathbf{b}(\mathbf{x}) = \begin{bmatrix} -2(x_1^3 - x_1) - \rho x_2 \\ -x_2 + 2\rho(x_1^3 - x_1) \end{bmatrix}$.
- `char chfield = 'y'`, a nonlinear vector field $\mathbf{b}(\mathbf{x}) = \begin{bmatrix} \rho x_2 - x_1(1 - x_1^2 - x_2^2) \\ -\rho x_1 - x_2(1 - x_1^2 - x_2^2) \end{bmatrix}$.

The ratio ρ of the magnitudes of the rotational and potential components of the vector field is defined by the global variable `double fac` with declaration in line 110.

`olim2D.c` and `olim2D_meas.c` have three running options specified in lines 31-34.

- Set `#define CH_SHOOT_MAP 'y'` if you want to compute the quasi-potential and shoot a MAP right after that.
- Set `#define CH_SHOOT_MAP 'n'` if you want only to compute the quasi-potential but do not want to shoot a MAP. You will be able to do it later.
- Set `#define CH_SHOOT_MAP 's'` if you want to shoot a MAP and have already computed the quasi-potential.

For now, we only compute the quasi-potential but do not shoot a MAP. So always set `#define CH_SHOOT_MAP 'n'` in line 34.

4. Output and plotting graphs in MATLAB

The computation terminates as soon as the boundary of the computational domain is reached at some point. The values of the computed quasi-potential are stored in the global variable `double g[NX*NY]`, where those not computed are equal to 10^6 (which plays the role of ∞ in the codes). The values of the exact quasi-potential are stored in the global variable `double Uexact[NX*NY]`.

Only the code `olim2D_meas.c` (modified for measurements) produces output files, with the parameters, the errors and CPU time for plotting graphs in MATLAB. The output files are saved under the subdirectory "measurements", which needs to be manually created under the current directory before running the C code. The names of the output files are set up in lines 908 and 928:

```
char fname[100];
sprintf(fname,"measurements/olim2D_chfield_%c_fac_%g_N_%li_testK.txt",chfield,fac,NX);
```

These files contain the update factor K ranging from 1 to 40, the two normalized numerical errors and CPU time for each specified vector field, ratio ρ, and mesh size, in the format:

```
K   Normalized ErrMax   Normalized ERMS   cputime of olim()
```

For plotting graphs in MATLAB, we experiment on the following three examples with mesh sizes `NX = NY = pow(2,p) + 1, p = 7,8,9,10,11,12` :

- Example 1: `char chfield = 'd'; double fac = 10.0;`
- Example 2: `char chfield = 'y'; double fac = 4.0;`
- Example 3: `char chfield = 'y'; double fac = 10.0;`

Once `olim2D_meas.c` produced all the output files, run the MATLAB script `plot_testK.m` to plot the normalized errors versus the update factor K and observe a guideline for choosing K. Then run the MATLAB script `plot_err_Kopt.m` to plot the CPU time versus the mesh size N, the normalized errors versus N, and the CPU time versus the normalized errors (efficiency), with the optimal choice of update factor K.

5. Changing mesh size, the parameters, and the vector field

The mesh size `NX`$\times$`NY` and the update factor `K` are specified in lines 26-30. Depending on the vector fields, we have different recommendations for choosing the value of the update factor K. With mesh sizes $N^2 = (2^p + 1)^2$, $p = 7, 8, 9, 10, 11, 12$,

- For linear vector field `chfield = 'l'`, we use the Rule-of-Thumb proposed in [2]:
$$K(N) = \texttt{round}[\log_2 N] - 3.$$
- For nonlinear vector fields `chfield = 'd'` and `chfield = 'y'`, we use a guideline in the table below (also written in commented lines 7-12 for your convenience):

N	129	257	513	1025	2049	4097
K	8	10	13	16	21	26

The computational domain is set up in the function `void param()` starting on line 220.

The index of the asymptotically stable equilibrium $\mathbf{x}^*$ with respect to which the quasi-potential will be computed is specified by the global variable `long Iindex` whose value is assigned in the function `void param()`.

If you want to add your own vector field, do the following steps:

(1) Pick some character to denote your vector field, for example, `'x'`, and change the value of the variable `chfield` in line 108 to `'x'`:
```
char chfield = 'x';
```
(2) In function `struct myvector myfield(struct myvector x)` starting on line 136, add `case 'x'` to the operator `switch`:
```
case 'x':
v.x = < insert your b_1(x.x, x.y) here >;
v.y = < insert your b_2(x.x, x.y) here >;
break;
```
(3) If the exact quasi-potential is available, add `case 'x'` to the operator `switch` in function `double exact_solution(struct myvector x)` starting on line 165:
```
case 'x':
return < insert your U(x.x, x.y) here >;
```
Also, add
```
|| chfield == 'x'
```
right before `)` when the indicator `ch_exact_sol` is specified in line 223.

(4) Calculate the Jacobian of your vector field and evaluate it at the asymptotically stable equilibrium $\mathbf{x}^*$ with respect to which you want to compute the quasi-potential. Let J denote the obtained Jacobian matrix: $J = \begin{bmatrix} J_{11} & J_{12} \\ J_{21} & J_{22} \end{bmatrix} = \begin{bmatrix} \partial_{x_1} b_1 & \partial_{x_2} b_1 \\ \partial_{x_1} b_2 & \partial_{x_2} b_2 \end{bmatrix}$. Matrix J is defined in `olim2D.c` in the global variable `struct mymatrix Amatrix`. In function `void makeAmatrix(char chfield)` starting on line 185, add `case 'x'` to the operator `switch`:

```
case 'x':
Amatrix.a11 = < insert your J_11 here >;
Amatrix.a12 = < insert your J_12 here >;
Amatrix.a21 = < insert your J_21 here >;
Amatrix.a22 = < insert your J_22 here >;
break;
```

(5) In function `void param()` starting on line 220, add `case 'x'` to the operator `switch` with the index of the mesh point at which the asymptotically stable equilibrium $\mathbf{x}^*$ is located and the limits of the rectangular computational domain:

```
case 'x':
Iindex = < insert the index of x* here >;
XMIN = ...  ; XMAX = ...  ;
YMIN = ...  ; YMAX = ...  ;
break;
```

Now compile and run the code!

References

[1] S. Yang, S. Potter, and M. Cameron, Computing the quasipotential for non-gradient SDEs in 3D, *submitted to Journal of Computational Physics*, 2018, arXiv: 1808.00562

[2] D. Dahiya and M. Cameron, Ordered Line Integral Methods for Computing the Quasi-potential, Journal of Scientific Computing, Springer, 2017

[3] https://www.math.umd.edu/∼mariakc/olimpackage.zip

[4] https://www.math.umd.edu/∼mariakc/olim3dqpotpackage.zip