C CODES IMPLEMENTING ORDERED LINE INTEGRAL METHODS

MARIA CAMERON AND DAISY DAHIYA

Contents

In this document, we provide a user guide for the C codes implementing Ordered Line Integral Methods for computing the quasi-potential in 2D. The description of the methods, the results of the numerical tests, and error analysis are given in [1].

1. Contents

The package contains four C-codes implementing Ordered Line Integral Methods for computing the quasi-potential in 2D:

- OLIM_midpoint.c (using the midpoint rule for line integrals),
- OLIM_trapezoid.c (using the trapezoid rule for line integrals),
- OLIM_simpson.c (using Simpson's rule for line integrals),
- OLIM_righthand.c (using the righthand rectangle rule for line integrals),

and two input files

- circle.txt, a set of 399 points equispaced on the unit circle $x^2 + y^2 = 1$,
- icurve_brusselator.txt, a set of 329 points lying on the limit cycle of the Brusselator with $a = 1$, $b = 3$.

We favor OLIM-midpoint.c because it has the best balance between the CPU time and accuracy. OLIM-trapezoid.c is almost as good. OLIM-simpson.c may produce even smaller numerical error on the same mesh sizes as OLIM-midpoint.c but requires larger CPU times. OLIM-righthand.c is the fastest method. It is provided mainly for comparison. It's numerical scheme is equivalent to the one in the OUM (the Ordered Upwind Method) implemented in the r-package QPot available at https://cran.r-project.org/web/packages/QPot/index.html All the OLIMs provided here are faster than the OUM.

2. How to use the C codes

2.1. **Compiling and Running.** We run the codes using the gcc C compiler available in the Command Line Tool. Any other C compiler should be applicable.

Open Terminal, change the directory to the one where you stored the files from the package, and type:

```
gcc OLIM_midpoint.c -lm -O3
```

Then type:

```
./a.out
```

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

```
in param()
in olim()
576605 (511 2) is accepted, g=9.9223e-01
cputime of olim() = 33.2306
NX = 1024, NY = 1024
errmax = 3.9602e-05, erms = 2.6238e-05
```

2.2. **Input and output.** The C codes are designed for computing the quasi-potential in 2D with respect to two kinds of attractors: asymptotically stable equilibria and asymptotically stable limit cycles. If you run any of them for computing the quasi-potential with respect to an asymptotically stable equilibrium, no input is required. If you run any of them for computing the quasi-potential with respect to a limit cycle, you need to provide a plain text file with the set of points more or less uniformly distributed along the limit cycle. Format of the input file:

$$\begin{matrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_M & y_M \end{matrix}$$

If the number of points M exceeds 399, defined in

```
#define NCURVEMAX 399
```

(line 18 of the codes), change NCURVEMAX to M or any reasonable number greater than M. The name of the input files is specified in the line

```
const char *ficurve_name = "circle.txt"; // input file with points of the initial curve
```

(approximately line 125). The default input file name is circle.txt. Change it to the name of your input file.

The codes produce two output files: Qpot.txt and stype.txt. The file Qpot.txt contains the computed quasi-potential on the $N_y \times N_x$ mesh in the format:

$$\begin{matrix} u_{11} & u_{12} & \dots u_{1N_x} \\ u_{21} & u_{22} & \dots u_{2N_x} \\ \vdots & \vdots & \vdots \quad \vdots \\ u_{N_y1} & u_{N_y2} & \dots u_{N_yN_x} \end{matrix}$$

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

The file stype.txt provides the information what kind of update, the one-point update or the triangle update, has produced the final value of the solution at each mesh point. The format of `stype.txt` is an $N_y \times N_x$ array corresponding to the array in Qpot.txt.
stype(i,j) = -1 means that the quasi-potential was not computed at the mesh point (i, j);
stype(i,j) = 0 means that the accepted value at the mesh point (i, j) was produced by the initialization procedure;
stype(i,j) = 1 means that the accepted value at the mesh point (i, j) was produced by the one-point update;
stype(i,j) = 2 means that the accepted value at the mesh point (i, j) was produced by the triangle update.

The names of the output files are specified in lines

```
const char *f_qpot_name = "Qpot.txt"; // output file with the quasipotential
const char *f_solinfo_name = "stype.txt"; // output file with the solution type
```

(approximately lines 126-127).

2.3. **Changing mesh size, update parameter, and the vector field.** The mesh size $N_y \times N_x$ and the update parameter K are specified in lines 15-17:

```
#define NX 1024
#define NY 1024
#define K 22
```

We recommend to pick the value of the update parameter K according the following rules-of-thumb:

The Rule-of-Thumb for OUM and OLIM-R. For an $N \times N$ mesh where $2^7 \le N \le 2^{12}$, pick

$$K(N) = \mathsf{round}[\log_2 N] - 3. \tag{1}$$

The Rule-of-Thumb for OLIM-MID, OLIM-TR, and OLIM-SIM. For an $N \times N$ mesh where $2^7 \le N \le 2^{12}$, pick

$$K(N) = 10 + 4(\mathsf{round}[\log_2 N] - 7). \tag{2}$$

The codes contain 5 options for choosing the vector field $\mathbf{b}(\mathbf{x})$ in the SDE $d\mathbf{x} = \mathbf{b}(\mathbf{x})dt + \sqrt{\epsilon}d\mathbf{w}$ for which the quasi-potential is computed. These options are specified by the global variable chfield in line approximately 110:

```
// choose the vector field b
const char chfield='l';
// Vector fields with asymptotically stable equilibrium
// chfield = 'l': b = [-2, -alin; 2*alin, -1][x; y]; (in the Matlab notations),
// analytic solution  U = 2x^2 + y^2
```

```
const double alin = 10.0;
// chfield = 'q' ->  Maier-Stein model;
// chfield = 'r' -> FitzHugh-Nagumo model with a = 1.2

// Vector fields with stable limit cycle
// chfield = 'b' ->  Brusselator;
// CHANGE const char *ficurve_name = "circle.txt"; to
// const char *ficurve_name = "icurve_brusselator.txt";
// chfield = 'c' -> analytic solution U = (1/2)(r^2-1)^2, l = (y,-x)
```

In order to add your own 2D vector field, do the following

(1) pick some character to indicate it, for example, set chfield = 'x'; (approximately line 110).

(2) In function struct myvector myfield(struct myvector x) (approximately line 131), add above the default:

```
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 you want to compute the quasi-potential with respect to an **asymptotically stable equilibrium** $(x0, y0)$, in function void param() (approximately line 178), add (above the default) the coordinates of the equilibrium point and the limits of the rectangular computational domain:

```
case 'x':
x_ipoint.x=< insert the value of x0 here >;
x_ipoint.y=0.< insert the value of y0 here >;
XMIN = ... ; XMAX = ... ;
YMIN = ... ; YMAX = ... ;
break;
```

If you want to compute the quasi-potential with respect to a **stable limit cycle** in function void param() (approximately line 178), add (above the default) the limits of the rectangular computational domain:

```
case 'x':
XMIN = ... ; XMAX = ... ;
YMIN = ... ; YMAX = ... ;
break;
```

(4) If you want to compute the quasi-potential with respect to an **asymptotically stable equilibrium**, in function int main(), modify the piece of code with switch operator (approximately lines 962-973) to

```
switch( chfield ) {
 case 'l': case 'q': case 'r': case 'x':
 ipoint();
```

```
break;
case 'c': case 'b':
initial_curve();
break;
default:
printf("Enter a correct value of the char variable chfield\n");
exit(1);
break;
}
```

If you want to compute the quasi-potential with respect to a **stable limit cycle**, in function int main(), modify the piece of code with switch operator (approximately lines 962-973) to

```
switch( chfield ) {
 case 'l': case 'q': case 'r':
 ipoint();
 break;
 case 'c': case 'b': case 'x':
 initial_curve();
 break;
 default:
 printf("Enter a correct value of the char variable chfield\n");
 exit(1);
 break;
 }
```

(5) If you want to compute the quasi-potential with respect to a **stable limit cycle**, create a .txt file (e.g. mycycle.txt) with the initial curve representing the stable limit cycle (see Section 2.2) and change the name of the input file to your mycycle.txt.

Now run the code!

2.4. **Visualizing the quasi-potential in MATLAB.** The most basic Change the directory in Matlab to the directory where the codes are and type in the command window:

```
u = load('Qpot.txt');
ind = find(u > 1e5);
u(ind) = NaN;
figure
imagesc(u)
```

References

[1] D. Dahiya and M. Cameron, Ordered Line Integral Methods for Computing the Quasi-potential, *submitted to Journal of Scientific Computing, Springer*, 2017, arXiv: