User's guide for StocDeltaN

- *field-space type* -

Yuichiro Tada[1,*]

[1] *Department of Physics, Nagoya University, Nagoya 464-8602, Japan*

(Dated: November 12, 2018)

I. CODE OVERVIEW

StocDeltaN is a powerful C++ package to analyze inflationary dynamics and calculate the power spectrum of curvature perturbations with use of the techniques of the stochastic-δN approach [1, 2]. We provide two types of StocDeltaN: one is the full phase-space formulation time without the slow-roll approximation, while the other is the field-space formulation omitting the degrees of freedom (d.o.f.) of inflaton momenta under the slow-roll approximation. Though the latter, which is labeled by "_conf" (meaning of "configuration". to be renamed?), sometimes wrong in models where e.g. the slow-roll conditions are violated for an instant, it is much more economic computationally thanks to the halved d.o.f. and often enough for leading order calculations. In this user's guide, we focus on this field-space type though the usage is naturally extended to the phase-space type.

StocDeltaN consists of two parts as "source" part containing all required numerical solvers which we provide and "main" part where users specify the inflationary model and the usage of several options like a plotting option with use of Python. Users can write the main code using various sample codes as references.

"Source" part in itself is divided into three parts as `JacobiPDE_conf.cpp`, `SRK32_conf.cpp`, and `StocDeltaN_conf.cpp`. `JacobiPDE_conf.cpp` and `SRK32_conf.cpp` implement the solver classes for partial differential equations (PDE) and stochastic differential equations (SDE) respectively. Overriding these classes, user can use the numerical solver for general problems beyond the inflationary system if want. `StocdeltaN_conf.cpp` overrides their classes and defines the specified class as the stochastic-δN solver. The main code specifies the inflationary model, overriding its functions defining e.g. potential, field-space metric, and so on, and then user can use its member functions to analyze the stochastic inflation.

* tada.yuichiro@e.mbox.nagoya-u.ac.jp

II. TUTORIALS

A. Prerequisties

- **C++ compiler** : We have checked the operation in GNU, Clang, and Intel compiler on Mac system. GNU compiler will be used by default. On Mac system, generally Clang compiler is preinstalled and automatically called instead of GNU compiler, so users need not to install other compilers by themselves. By modifying Makefile, one can change the used compiler if wants.

- **Make** : For an easy compilation, StocDeltaN employs Make which is a compilation manager. On Mac system, it is easy to introduce GNU Make by installing Command Line Tools of Xcode.

- **OpenMP (optional)** : The modern concept of processor design places importance on parallelization over multicore rather than processing power on each single core. Multicore processors are implemented even on personal computers and parallelization gets much closer to end users. StocDeltaN employs automatic parallelization with use of OpenMP. Many representative compilers like GNU or Intel's one precontain OpenMP, and therefore users can make use of its parallelization simply by validating OpenMP option (e.g. -fopenmp for GNU compiler) in Makefile. Clang compiler preinstalled on Mac may not support OpenMP, then OpenMP option is commented out in a sample Makefile. But we strongly recommend the usage of OpenMP to bring out the full performance of StocDeltaN particularly in calculation on cluster computers.

- **Python & Matplotlib (optional)** : Though StocDeltaN exports all relevant data as DAT files, one can also use the automatic plotting system with use of Matplotlib which is Python's plotting library. Piping data to python, StocDeltaN can plot the obtained results if Python and Matplotlib have been installed on a users' machine. Mac system usually preinstalled Python, and therefore one can easily install Matplotlib as well by using pip, e.g.

```
$ pip install matplotlib
```

B. First Exercise

Practice makes perfect. So let us start by downloading the source codes from our GitHub page https://github.com/NekomammaT/StocDeltaN_dist and play with attached samples. Using Make in the *sample* directory completes the compilation of the sample codes:

```
$ cd sample
$ make
```

By default the double chaotic model is compiled. Then running the executable file *double_chaotic_conf*, one obtains some results like as follows.

```
$ ./double_chaotic_conf
[xi, xf, xmin, xmax]:
[13, 3.54334e-08, -5.06876e-07, 13]
[13, 1.40077, 1.40077, 13]

N = 84.01
Vi = 8.5543209876543225e-09,  Vf = 1.2111996327249413e-12
0.190044 sec.
```

If you could do this without any error, the machine environment is properly set up and you are ready to execute a main calculation.

One edits the following parts of *double_chaotic_conf.cpp*. Simply by commenting out `sdn.sample()` and validating `sdn.solve()` instead as indicated in List. 1, one can fully solve the system. Re-Making and running *double_chaotic_conf*, one obtains the analyzation results in *Mn_double_chaotic_conf.dat*, *traj_double_chaotic_conf.dat*, and *calP_double_chaotic_conf.dat*, which contain the data as shown in Lists. 2–4.

Listing 1. *sample/double_chaotic_conf.cpp*

```
77      //sdn.sample(); // obtain 1 sample path
78      //sdn.sample_plot(); // plot that sample path
79
80      sdn.solve(); // solve PDE & SDE and obtain power spectrum
81      //sdn.f1_plot(); // show plot of <N>
82      //sdn.g2_plot(); // show plot of <delta N^2>
83      //sdn.calP_plot(); // show plot of power spectrum of zeta
```

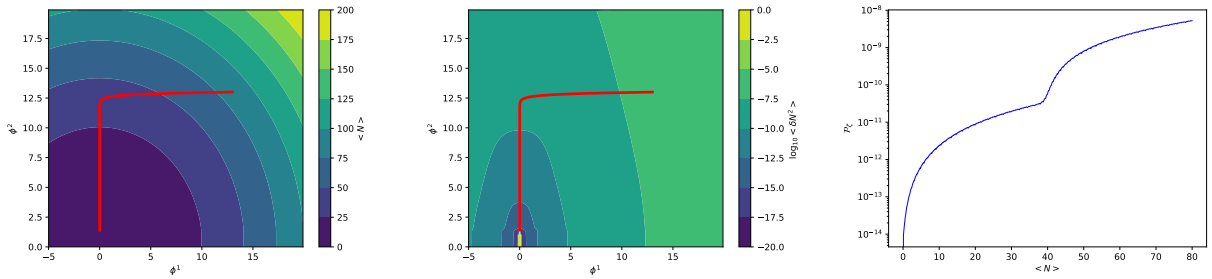

FIG. 1. *N_double_chaotic_conf.pdf*, *dN2_double_chaotic_conf.pdf*, and *calP_double_chaotic_conf.pdf* from left to right. Red lines in the left and middle panels show one realization of sample paths.

Listing 2. *Mn_*⟨model name⟩*.dat* : contour data of $\mathcal{M}_n = \langle \mathcal{N}^n \rangle$ and $\mathcal{C}_2 = \langle \delta\mathcal{N}^2 \rangle$

```
ϕ¹  ϕ²  ···  M₁  C₂
```

Listing 3. *traj_*⟨model name⟩*.dat* : trajectory data of one sample path

```
N  ϕ¹  ϕ²  ···
```

Listing 4. *calP_*⟨model name⟩*.dat* : data related to curvature perturbation

```
⟨N⟩  ⟨δN²⟩  P_ζ
```

If you have installed Python and Matplotlib already, the plotting option is quite useful. Validating `sdn.f1_plot()`, `sdn.g2_plot()`, and `sdn.calP_plot()` shown in List. 1, StocDeltaN pipes the data to Python and plots contours of $\mathcal{M}_1 = \langle \mathcal{N} \rangle$ and $\mathcal{C}_2 = \langle \delta\mathcal{N}^2 \rangle$ and the power spectrum of curvature perturbations $\mathcal{P}_\zeta$ as shown in Fig. 1. The plots are also saved as *N_double_chaotic_conf.pdf*, *dN2_double_chaotic_conf.pdf*, and *calP_double_chaotic_conf.pdf*.

To switch the analyzed model, one modifies *Makefile* in the *sample* directory. The `MODEL` parameter in *Makefile* as can be seen in List. 5 controls the analyzed model and one can calculate other models by changing this parameter. We preprovide other three models, *chaotic_conf*, *hilltop_conf*, and *hybrid_conf*, as samples. One can use preferred C++ compiler by changing `CXX` parameter. The default setting is GNU compiler `g++` (On Mac system, it basically calls Clang compiler instead which may not support OpenMP. That is why we comment out OpenMP option `-fopenmp` by default). If you use a compiler supporting OpenMP, you may use it by validating the OpenMP option `-fopenmp` (or `-qopenmp` for Intel compiler).

Listing 5. *sample/Makefile*

```
MODEL = double_chaotic_conf
SOURCE = ../source
PHASECONF = conf

CXX := g++
CXXFLAGS := -std=c++11 -O3 #-fopenmp
```

C. Go beyond samples

Of course the final goal is not playing with sample codes but solving your models. Though ultimately the understanding of the whole code structure is required to get a full command of StocDeltaN, we recommend to take advantage of sample codes as much as possible. That is, if you want to solve single-field models, *chaotic_conf.cpp* or *hilltop_conf.cpp* would be helpful, while *double_chaotic_conf.cpp* or *hybrid_conf.cpp* can be used for two-field models. To adopt your model to StocDeltaN, you change the code in mainly two sences: one is the inflaton's system described by the Lagrangian, and the other is the numerical parameters used for practical calculations. We guide the readers in this way below.

1. Lagrangian

StocDeltaN supports general multi-scalar models in general relativity. Such models are parameterized the following scalar part Lagrangian.

$$\mathcal{L} = -\frac{1}{2}G_{IJ}(\phi)\partial_\mu\phi^I\partial^\mu\phi^J - V(\phi). \tag{1}$$

Here I and J label the scalar fields and G_{IJ} represents the inflaton's field space metric which can be curved in general.

This Langrangian is set in StocDeltaN by the corresponding functions in a main code represented by e.g. *double_chaotic_conf.cpp* as shown in List. 6. Each function represent:

- `V(X)` : the inflaton potential $V(\phi)$.
- `VI(X,I)` : its derivative $\partial_{\phi^I}V(\phi)$.
- `metric(X,I,J)` : the field space metric $G_{IJ}(\phi)$.

- `inversemetric(X,I,J)`: its inverse $G^{IJ}(\phi)$.
- `affine(X,I,J,K)` : the Christoffel symbol corresponding to the field space metric, $\Gamma^I_J K(\phi) = \frac{1}{2} G^{IL}(G_{JL,K} + G_{KL,J} - G_{JK,L})$.

double_chaotic_conf.cpp studies the double mass-term inflation with a flat field space as

$$V = \frac{1}{2}M^2\phi^2 + \frac{1}{2}m^2\psi^2, \qquad G_{IJ} = \delta_{IJ}. \tag{2}$$

In the code the list `X` represents the scalar fields as `X[0]` $= \phi$ and `X[1]` $= \psi$, then the functions are defined as shown in List. 6. The potential parameters M and m are defined at the top of the code as `MPHI` and `MPSI` to be easily modified. Therefore one can adopt the model interested simply by rewriting this part.

Listing 6. *sample/double_chaotic_conf.cpp*

```
20 // ---------- potential parameter ----------
21 #define MPHI (1e-5)
22 #define MPSI (MPHI/9.)
23 // -----------------------------------------
```

```
93  // ---------- Lagrangian params. and diff. coeff.  X[0]=phi, X[1]=psi ----------
94
95  double StocDeltaN::V(vector<double> &X)
96  {
97    return 1./2*MPHI*MPHI*X[0]*X[0] + 1./2*MPSI*MPSI*X[1]*X[1];
98  }
99
100 double StocDeltaN::VI(vector<double> &X, int I) // \partial_I V
101 {
102   if (I == 0) {
103     return MPHI*MPHI*X[0];
104   } else {
105     return MPSI*MPSI*X[1];
106   }
107 }
108
109 double StocDeltaN::metric(vector<double> &X, int I, int J) // G_IJ
110 {
111   if (I == J) {
112     return 1;
113   } else {
114     return 0;
```

```
  }
}

double StocDeltaN::inversemetric(vector<double> &X, int I, int J) // G^IJ
{
  return metric(X,I,J);
}

double StocDeltaN::affine(vector<double> &X, int I, int J, int K) // \Gamma^I_JK
{
  return 0;
}
```

2. *lattice size*

The other important parameter for StocDeltaN is the lattice size defining the inflationary region. StocDeltaN adopts the orthogonal box with arbitrary variable lattice steps as shown in Fig. 2. The box can include the non-inflationary region where the corresponding energy density is lower than the threshold ρ_c shown by the blue shade in Fig. 2, other than the yellow inflationary region Ω. The box is parametrized two-dimensional list containing the field values of lattice sites in each scalar direction, i.e.,

```
{{φ0, φ1, φ2, ···, φmax}, {ψ0, ψ1, ψ2, ···, ψmax}}
```

in the case of Fig. 2.

In *double_chaotic_conf.cpp*, the box boundary is defined by `PHIMIN`, `PHIMAX`, `PSIMIN`, and `PSIMAX` with the constant lattice steps `HPHI` and `HPSI`, declared at the top of the code as shown in List. 7. The concrete two-dimensional list is created in the latter part as `sitepack`.

Listing 7. *sample/double_chaotic_conf.cpp*

```
// ---------- box size & step h ------------
#define PHIMIN -5
#define PHIMAX 20
#define PSIMIN 0
#define PSIMAX 20
#define HPHI (1e-1)
#define HPSI (1e-1)
// -----------------------------------------
```

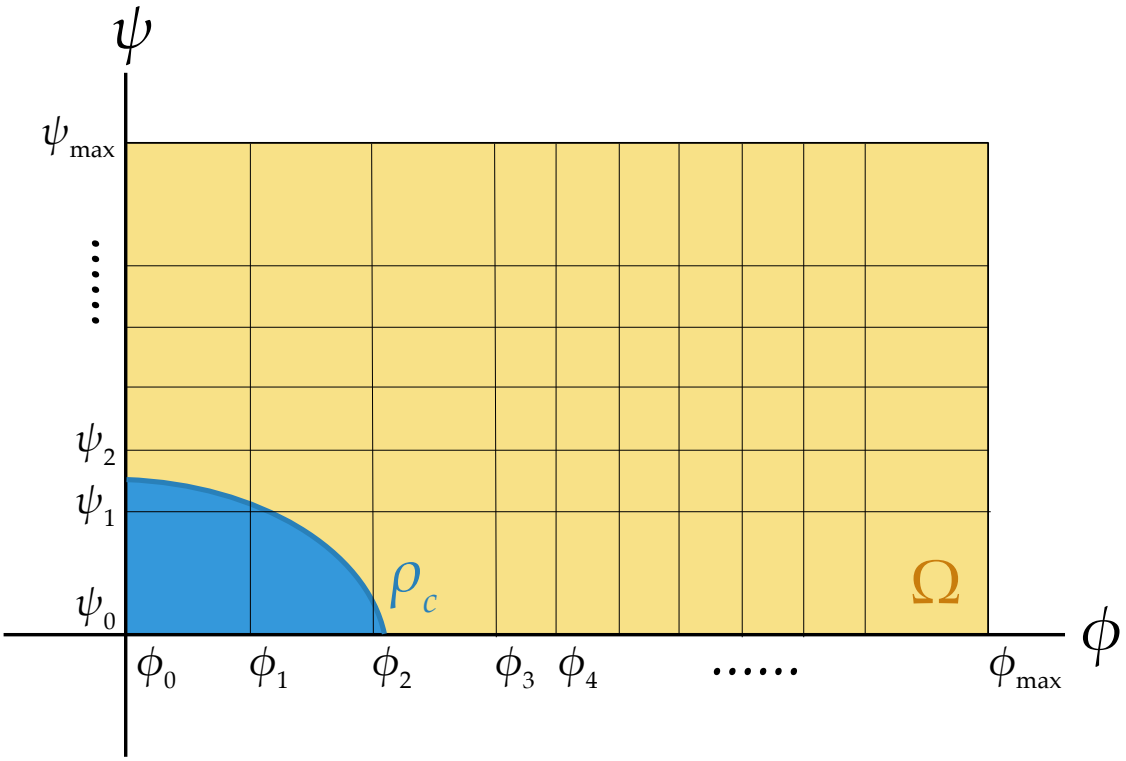

FIG. 2. Schematic image of the calculable lattice in StocDeltaN. Users declare each field value ϕ_0, ϕ_1, ϕ_2, $\cdots$, $\phi_{\max}$, and ψ_0, ψ_1, ψ_2, $\cdots$, $\psi_{\max}$.

```
52      // ---------- set box step h ---------------
53      double h = HPHI, sitev = PHIMIN;
54      vector<double> site;
55      vector< vector<double> > sitepack;
56      while (sitev <= PHIMAX) {
57          site.push_back(sitev);
58          sitev += h;
59      }
60      sitepack.push_back(site);
61      site.clear();
62
63      h = HPSI, sitev = PSIMIN;
64      while (sitev <= PSIMAX) {
65          site.push_back(sitev);
66          sitev += h;
67      }
68      sitepack.push_back(site);
69      site.clear();
70      // ----------------------------------------
```

One might prefer the logarithmic lattice to the constant step size. In such a case, it is useful to determine the step size by the field value itself. *hybrid_conf.cpp* employs this procedure in ψ's direction. In line 70 shown in List. 8, the step size in ψ's direction is determined by its ψ-value itself with a fraction `HPSIOPSI` and the lower bound `HPSIMIN`. Therefore the lattice step size becomes smaller for the smaller value of $|\psi|$, resulting in the logarithmic scale.

Listing 8. *sample/hybrid_conf.cpp*

```
6  // ---------- box size & step h ------------
7  #define PHIMIN 0.1409
8  #define PHIMAX 0.142
9  #define PSIMIN -(1e-3)
10 #define PSIMAX (1e-3)
11 #define HPHI (1e-5)
12 #define HPSIOPSI (1e-2) // hpsi/|psi|
13 #define HPSIMIN (1e-10)
14 // ----------------------------------------
```

```
57     // ---------- set box step h ---------------
58     double h = HPHI, sitev = PHIMIN;
59     vector<double> site;
60     vector< vector<double> > sitepack;
61     while (sitev <= PHIMAX) {
62         site.push_back(sitev);
63         sitev += h;
64     }
65     sitepack.push_back(site);
66     site.clear();
67
68     sitev = PSIMIN;
69     while (sitev <= PSIMAX) {
70         h = max(fabs(sitev)*HPSIOPSI,HPSIMIN);
71
72         site.push_back(sitev);
73         sitev += h;
74     }
75     sitepack.push_back(site);
76     site.clear();
77     // ----------------------------------------
```

3. other parameters

Other parameters are set at the top of the code and directly called for the declaration of the StocDeltaN class except for the initial condition for sample trajectories (`PHIIN` and `PSIIN` in this case) as shown in List. 9.

Listing 9. *sample/double_chaotic_conf.cpp*

```
4  #define MODEL "double_chaotic_conf" // model name
```

```
15 // ---------- for PDE ----------------------
16 #define MAXSTEP 100000 // max recursion
17 #define TOL 1e-10 // tolerance
18 // -----------------------------------------
```

```
25 #define RHOC (MPSI*MPSI) // end of inflation
26
27 // ---------- for SDE ----------------------
28 #define RECURSION 100 // recursion for power spectrum
29 #define PHIIN 13 // i.c. for phi
30 #define PSIIN 13 // i.c. for psi
31 #define TIMESTEP (1e-2) // time step: delta N
32 // -----------------------------------------
33
34 // ---------- for power spectrum -----------
35 #define DELTAN 0.1 // calc. PS every DELTAN e-folds
36 #define NMAX 80 // calc. PS for 0--NMAX e-folds
37 // -----------------------------------------
```

```
72     vector<double> xi = {PHIIN,PSIIN}; // set i.c. for sample paths
73
74     StocDeltaN sdn(MODEL,sitepack,RHOC,xi,0,MAXSTEP,TOL,RECURSION,
75         TIMESTEP,NMAX,DELTAN); // declare the system
```

III. CODE STRUCTURES

[1] Tomohiro Fujita, Masahiro Kawasaki, Yuichiro Tada, and Tomohiro Takesako, "A new algorithm for calculating the curvature perturbations in stochastic inflation," JCAP **1312**, 036 (2013), arXiv:1308.4754 [astro-ph.CO].

[2] Vincent Vennin and Alexei A. Starobinsky, "Correlation Functions in Stochastic Inflation," Eur. Phys. J. **C75**, 413 (2015), arXiv:1506.04732 [hep-th].