The software is part of the SISDECS project, #52 [Hellenic Foundation for Research and Innovation (H.F.R.I.)](http://www.google.com/url?q=http%3A%2F%2Fwww.elidek.gr%2Fen%2Fhomepage%2F&sa=D&sntz=1&usg=AOvVaw2amYBB4lsyLTzjehANS-SX).

This software provides python modules for inference methods for stochastic differential equations (SDEs). It is build on the benchmark SDEs (1) 1-dimensional Ornstein-Uhlenbeck (2) 1-dimensional Jacobi and (3) 2-dimensional Ornstein-Uhlenbeck processes.

1. For an 1D OU processes the files main\_mle\_sim:
2. Reads a data file or creates data using the function sp\_generator\_ou(time steps, samples, t\_min, t\_max, theta) from the sample generator module.
3. Creates a sparser than the initial data-set according to a chosen time step using the function step\_data.step\_data\_1D(data X, data time, number of initial time steps, number of sparser time steps grid, samples)
4. Solves the MLE problem using the likelihood\_test module:

* The function likelihood\_test.mle( initial guess theta, mean value at t=0, variance at t=0, data, theta, time steps, moments, LT) for LT = 1 and moments =1 corresponds to the exact likelihood calculated by the analytical solution of the OU moment equations . This function uses the function mean\_ou(t, x0, theta) from the moments\_ou module to obtain the mean value and the corr\_ou1(theta[0],theta[2], time steps) from the corr\_ou module covariance matrix.
* The function likelihood\_test.mle for LT = 1 and moments =2 corresponds to the exact likelihood calculated by the numerical solution of the OU moment equations. The Python function solve\_ivp is used to solve the system of moment equations for the mean value and the variance calling the ou\_moment\_f function of the moments\_ou module. The solution of the two moments equations provides the diagonal elements of the covariance matrix. The solve\_ivp function is then used again to find the off-diagonal elements using the variance of each row as initial value. The moment equation for the covariance is the ou\_moment\_cc function of the moments\_ou module.
* The function likelihood\_test.mle for LT = 2 and moments = 5 corresponds to the Euler pseudo likelihood when this is given by the local Gaussian approximation (Eq. 1.17 of the report) and likelihood\_test.sigma\_est(data, time) gives an estimation consistent estimator of the variance .
* The function likelihood\_test.mle for LT = 3 and moments = 5 corresponds to the Eulerian transition probability (Milstein) (Eq. 1.19 of the report) .
* The function likelihood\_test.sim\_mle(data, thetav, time steps, intermediate points , paths for the MC simulation of the intermediate points) corresponds to likelihood function as this is obtained by the simulated likelihood method. For each sample and each time step the transition pdf is calculated by calling the function OU\_Pdf.sl\_sim(Data value at t\_n, t,Data value at t\_(n-1) , t\_(n-1), theta, intermediate points , paths for the MC simulation of the intermediate points, sample path index).
* The function likelihood\_test.sim\_mle\_gb(data, thetav, time steps, indermidiate points , paths for the indermidiate points) (data, thetav, time steps, intermediate points , paths for the MC simulation of the intermediate points) corresponds to likelihood function as this is obtained by the Brownian Bridge method. For each sample and each time step the transition pdf is calculated by calling the function OU\_Pdf. sl\_sim\_gb\_log( Data value at t\_n, t,Data value at t\_(n-1) , t\_(n-1), theta, intermediate points , paths for the MC simulation of the intermediate points, sample path index).

1. Computes a confidence interval for the obtained parameters. The confidence integral is obtained by calculating the fisher matrix (I) that corresponds to the considered likelihood function(mle\_loc) for the obtained solution (b.x). This is realized by calling the fisher\_matrix(mle\_loc, b.x) function of the fisher\_matrix module that computes the Hessian using the function numdifftools.Hessian of the Python package numdifftools.Then the 96%, 98% and 98% CI are calculated by calling the fisher\_matrix.fisher\_matrix\_int\_est(b.x, I, percentage) function.

1. For an 1D Jacobi processes the file main\_Jacobi:
2. Reads a data file or create data using the function sp\_generator\_Jacobi (time steps, samples, t\_min, t\_max, theta) from the sample generator module.
3. Create a sparser than the initial data-set according to a chosen time step
4. Solves the MLE problem using the likelihood\_test module:

* The function likelihood\_test.mle( initial guess theta, mean value at t=0, variance at t=0, data, theta, time steps, moments, LT) for LT = 1 and moments =3 corresponds to the exact likelihood by numerical solution of the moments equation from the original Jacobi SDE
* The function likelihood\_test.mle( initial guess theta, mean value at t=0, variance at t=0, data, theta, time steps, moments, LT) for LT = 1 and moments = 4 moment equations from the Lamperti transformed Jacobi SDE,

1. For a 2D OU processes the file main\_2d reads:
2. Reads or Creates data of a 2D OU process function sp\_generator\_2d time steps, samples, t\_min, t\_max, theta, sigma,mean\_m, rho from the sample generator module.
3. Creates a sparser than the initial data-set according to a chosen time step and may also add noise to the created data.
4. Use the function mle\_2d(x0, data, parameters, tn) to compute the likelihood function by the analytical solution of the moment equations.