Computer_Homework3

Generated by Doxygen 1.9.1

Chapter 1

Computer Homework 3

Solve Kepler problem via Monte Carlo Metropolis method described in `Entropy 2020, 22(9), 916`

1.1 Requirements

To install this program, you should have

- C++ compiler like g++ (should support c++14 standard)
- gnu make or cmake

1.2 Installation

- gnu make
  1. modify make.inc file
  2. Type make, then you can see hw3 executable file in bin directory
  3. [optional] type make test to test your compilation
- cmake
  1. make build directory
  2. go to build directory and type cmake .. -Doption_flag=option
     - PRECISION_LEVEL level of precison 0: float, 1: double
     - MONITOR whether or not monitor optization process 0: no, 1: yes
     - PATH_TYPE type of path bezier: bezier curve, fourier: fourier function
  3. Type make then you can see hw3 executable in build directory

1.3 How To Use

Execute hw3 then, it will interactively read

- inital condition
- tolerance for the action integral
- number of sine and cosine will be used to approximate the path
- number of points to evaluate the minimized path
- step size
- parameter which controls acceptance of move
- number of iteration
- output file names

Then it computes and saves solution to file. You can plot the result using usual plotting software like gnuplot

1.4 How To plot saved monitor file

If you set -DMONITOR=1 in cmake build process, after execuation of hw3 program, you will get c++ binary file for the monitoring of the optimization process. If you also set -DPRECISION_LEVEL=0, datatype of such binary file would be float. Otherwise, datatype would be double. See libmcm::mcm::optimize for the detailed information of the format of such file. Anyway, If you have python3 and numpy and matplotlib module installed in it, then you can plot saved monitor file using plot_monitor_float.py (if -DPRECISION_LEVEL=0) or plot_monitor_double.py (if -DPRECISION_LEVEL=1) script located in monitor folder.

1.5 Copyright

1.6 License

Chapter 2

Evaluation of Action

The action S is defined as following

$$S = \int_{t_0}^{t_f} L(t, x(t), x'(t))\mathrm{d}t \tag{2.1}$$

, where $L(t, x(t), x'(t))$ is Lagrangian. In general, such integral could not be calculated analytically. To evaluate the action integral accurately, I use adpative gauss-kronrod quadrature method (method: 0) or tanh-sinh quadrature method (method: 1).

2.1 Gauss-Kronrod quadrature

In general, to numerically evaluates the integral

$$I = \int_a^b f(t)\mathrm{d}t \tag{2.2}$$

, we first approximate integrand $f(t)$ by polynomial. Once $f(t)$ is approximated by polynomial $P(t)$ then integration of $P(t)$ from a to b can be done analytically. There are several method to approximate integrand $f(t)$ by polynomial. Intuitive and simple method is using interpolation polynomial of order n with $n+1$ equally spaced points. Then we get Newton-Cots formula. However, such simple method suffers Runge phenomenon. So, high order Newton-Cots method is not accurate as we expected. Another method is using orthogonal polynomial to approximate integrand $f(t)$. In gauss-quadrature method, integrand $f(t)$ is approximated by Legendre polynomials. Then the integration I is approximated to

$$I = \int_a^b f(t)\mathrm{d}t \approx \sum_{i=1}^{n} w_i f(x_i) \tag{2.3}$$

, with abscissa

$$x_i = \frac{a+b}{2} + \eta_i \frac{b-a}{2} \tag{2.4}$$

, where η_i is the root of n th Lagendre polynomial $P_i(t)$ and weight

$$w_i = \frac{b-a}{(1-\eta_i^2)P'_n(\eta_i)^2} \tag{2.5}$$

Then such approximation is exact up to the polynomial of order $2n-1$ and order of error is known to be $O((b-a)^{2n+1})$ see `Kahaner, Moler & Nash 1989`. So, if we subdivide the interval by m then the error decreases by the order $O(m^{-2n-1})$. Kronrod improves gauss quadrature method by adding additional $n+1$ points to gauss absicca and introduce new weights to make the approximation is exact up to the polynomial of degree

$3n-1$. Denote x_i and w_i gauss absicca and weight and x'_i and w'_i kronrod absicca and weight. Let G_n and K_n be

$$G_n = \sum_{i=1}^{n} w_i f(x_i) \tag{2.6}$$

$$K_n = \sum_{i=1}^{2n+1} w'_i f(x'_i) \tag{2.7}$$

Then we can estimate error using difference between gaussian and kronrod quadrature $|G_n - K_n|$. Piessens suggests that error should be estimated by

$$E = (200 \cdot |G_n - K_n|)^{3/2} \approx 2829 \cdot (|G_n - K_n|)^{3/2} \tag{2.8}$$

2.2 Tanh-Sinh quadrature

Instead of approximate integrand $f(t)$ by polynomial, tanh-sinh quadrature exploits exponential convergence of trapeziodal rule when integrand decreases exponentially. To make integrand decay rapidly, consider following coordinate transformation.

$$t = \frac{a+b}{2} + \frac{b-a}{2}\tanh\left(\frac{\pi}{2}\sinh(u)\right) \tag{2.9}$$

Then integral I is

$$I = \int_a^b f(t)\mathrm{d}t = \frac{b-a}{2}\int_{-\infty}^{\infty} \frac{\frac{\pi}{2}\cosh u}{\cosh^2\left(\frac{\pi}{2}\sinh u\right)} f(t(u))\mathrm{d}u \tag{2.10}$$

Due to rapidly decreasing $t'(u)$ term, $t'(u)f(t(u))$ decays exponentially. Thus, if we approximate I as

$$I = \int_{-\infty}^{\infty} t'(u)f(t(u))\mathrm{d}u \approx I_h = h\sum_{i=-N}^{i=N} t'(ih)f(t(ih)) \tag{2.11}$$

, where h is step size. Then error of such approximation decays exponentially on not only N but also h. In finite precision system, N is taken to the smallest non-negative integer such that $|t'(Nh)f(t(Nh))| < \mathrm{eps}^2$, where eps is machine epsilon or practically, $|ht'(Nh)f(t(Nh))| < \mathrm{eps}\cdot(I_h + \mathrm{eps})$. Due to the double exponential convergence of tanh-sinh method, error of numerical integration with step size h can be estimated by

$$E = |I_{2h} - I_h| \tag{2.12}$$

when $h \leq 1/2$. Thus in practice, start from $h = 1$, evaluates I_h and estimates error using above equation (2.12) , then divide h by half. Repeats above step until estimated error less than tolerance. Tanh-sinh can handle integrand which has singularities at end points. However It is slower than guass-kronrod quadrature method when integrand is smooth and has no end points singularities. Therefore I set gauss-kronrod method with order 31 to default method for evaluation of action integration.

2.3 Convergence Criteria

Define L_1 norm of integrand $f(t)$ as

$$L_1 = \int_a^b |f(t)|\mathrm{d}t \tag{2.13}$$

and denote relative tolerance r then numerical integration is converged when estimated error E satisfies

$$E < r \cdot L_1 \tag{2.14}$$

However if relative tolerance r is comparable or smaller than machine epsilon, above convergence condition could not meet. In that case, numerical integration is stopped when estimated error E reaches numerical turncation limit.

2.4 Reliability

Reliability of such two integration method is summerized in `test.txt` file located in test sub directory.

Chapter 3

Approximation of Path

3.1 Fourier function

To approximate path, I define following fourier function.

$$\phi(t) = \sum_{i=0}^{N_f-1} c_{2i} \sin\left(\frac{2(i+1)\pi}{T}t\right) + c_{2i+1}\cos\left(\frac{2(i+1)\pi}{T}t\right) \tag{3.1}$$

, where N_f is the number of sine and cosine function to add, T is the period of the fourier function $\phi(t)$, and c_i are coefficients. To match the boundary condition, I introduce adder a and scaler s. Then the fourier path $\Psi(t)$ is defined as following

$$\Psi(t) = a + s\phi(t) \tag{3.2}$$

, where a, s are adder and scaler to match boundary condition and $\phi(t)$ is the fourier function (3.1) . Using adder and scaler, we can confine coefficients of fourier function c_i to $[-1, 1]$. When path $p(t)$ with initial time t_i and finial time t_f is given, then we can extend such path $p(t)$ to a periodic function $\hat{p}(t)$ of the period $2(t_f - t_i)$. If we set the period of fourier function used by fourier path to $2(t_f - t_i)$ then fourier path $\Psi(t)$ would approximate $\hat{p}(t)$. Since $\hat{p}(t) = p(t)$ when $t \in [t_i, t_f]$, $\Psi(t)$ also approximates $p(t)$ when $t \in [t_i, t_f]$. However if $t \notin [t_i, t_f]$, $\Psi(t)$ could not approximate $p(t)$, in general. This is due to the inconsistency of $\hat{p}(t)$ and $p(t)$ at $t \notin [t_i, t_f]$.

3.2 Bezier Curve

Bernstein proves the Weierstrass approximation theorem by showing

$$f_n(t) = \sum_{i=0}^{n} f\left(\frac{i}{n}\right)\binom{n}{i} t^i (1-t)^{n-i} \tag{3.3}$$

uniformly converges to continuous function $f(t)$ on $[0, 1]$. Thus, we can approximate path using Bezier Curve.

$$\phi(t, \{c_0, \ldots, c_n\}) = \sum_{i=0}^{n} c_i \binom{n}{i} t^i (1-t)^{n-i} \tag{3.4}$$

, where n is the order and c_i are control points of Bezier Curve. Note that Bezier curve has following properties.

$$\phi(0) = c_0 \tag{3.5}$$

$$\phi(1) = c_n \tag{3.6}$$

$$\phi(t, \{c_0, \ldots, c_n\}) = (1-t)\phi(t, \{c_0, \ldots, c_{n-1}\}) + t\phi(t, \{c_1, \ldots, c_{n-1}\}) \tag{3.7}$$

$$\phi'(t, \{c_0, \ldots, c_n\}) = n\phi(t, \{c_1 - c_0, \ldots, c_n - c_{n-1}\}) \tag{3.8}$$

Bezier curve can be evaluated directly, however such method is known to numerically unstable. So, I use more numerically stable but slower De Casteljau's algorithm which exploits recurrence property (3.7) to evaluate Bezier curve. However Bezier curve is only defined in $[0, 1]$. So, I define bezier_path surjectived with boundary condition $\Psi(t_i) = p_0$ and $\Psi(t_f) = p_1$ as

$$\Psi(t) = s\phi\left(\frac{t - t_i}{t_f - t_i}, \{c_0, \ldots, c_{n-1}, \frac{p_1}{p_0}c_0\}\right) \tag{3.9}$$

, where s is the scaling parameter used to match boundary codition. Then $\Psi(t)$ is defined in $[t_i, t_f]$ and we can bound control points c_i in $[0, 1]$.

Chapter 4

Monte Carlo Metropolis method

Details for the Monte Carlo Metropolis are described in `Entropy 2020, 22(9), 916`

4.1 Brief description

Briefly, the Monte Carlo Metropolis moves intial guess c by step_size via random walk. Denote moved guess c'. Then calculates acceptance ratio $\exp(-\lambda \Delta S)$, where $\Delta S = S_{c'} - S_c$. Next sample real number $r \in [0, 1]$ from the uniform distribution. If r is less than acceptance ratio then accepts move; set $c = c$' If r is greater than acceptance ratio then rejects move; set $c = c$ It repeats above step by n_iter times.

4.2 Target distribution

In Monte Carlo Metropolis method described in `Entropy 2020, 22(9), 916`, for large iteration number n_{iter}, the distribution $f_\lambda(S)$ converges to $g_\lambda(S)$, where

$$g_\lambda(S) = \begin{cases} p(S) \exp\left(-\lambda(S - S_{min})\right) & \text{,if } S \geq S_{min} \\ 0 & \text{,otherwise} \end{cases} \tag{4.1}$$

Due to degeneracy factor $p(S)$, in general, the most likely path differs from minimum action path. However, if we increase λ to penalize such discrepency between most likely action and minimum action, the most likely path would more close to minimum action path. Although increase λ greater the chance to find minimum action path, too large λ would make markov chain stuck. (i.e. low acceptance ratio) To deal with such problem we could decrease step↵ _size to lower the difference of action ΔS between each moves. The life is not as easy as you think it is. If you lower the step_size then you need more number of iteration to converge distribution. Therefore you should vary pair of step_size and λ to find the minimum action path with the fewer number of iteration.

4.3 Pseudo Code

1. intialize mcm class
   (a) set initial condition
   (b) set relative tolerance of the action integration see Evaluation of Action
   (c) set basic configuration for basis function used to approximate path.

   (d) set lagrangian $L(t, x(t), x'(t))$.

2. set initial guess, iteration number n_iter, step size and λ.
3. evaluate action and assign accept_action, min_action to such action
4. assign accept_guess and min_guess to initial guess
5. initialize i and n_accept to 0.
6. check $i <$ n_iter.
   (a) If yes, continue
   (b) If no, go to 15.
7. sample path via random walk at accept_guess see Random Walk to get detailed infomation
8. check vaildity of path
   (a) If path is vaild, update tmp_guess to that path and raise i by 1.
   (b) If path is not vaild, go to 7.
9. evaluate action at that path
10. compute difference of action $\Delta S =$ tmp_action $-$ min_action
11. Sample real number R from the uniform distribution which ranges 0 to 1.
12. Check $R < \exp(-\lambda \Delta S)$.
    (a) If yes, accept move. Update accept_action and accept_guess increase n_accept and i by 1.
    (b) If no, reject move. increase i by 1 and go to 6.
13. Check if accept_action $<$ min_action.
    (a) If yes, update min_action and min_guess to accept_action and accept_guess, respectively.
    (b) If no, do nothing.
14. Go to 6.
15. Finish iteration and report number of accepted move num_accept and acceptance ratio (num_accept/num↩ _iter)

After optimization, you can evaluate path at given points via min_eval method of mcm class and get coefficients of minimum path using get_min_coeff method.

4.4 Random Walk

To generate random walk, I use normal distribution whose mean and standard deriviation is 0 and step_size respectively. To move guess, Δc_i is sampled via normal distribution then add Δc_i to guess c_i. Moreover since guess c_i is confined to -1 from 1, if moved guess $c'_i = c_i + \Delta c_i$ is above 1 or below -1, then $c'_i = 1$ when $c'_i > 1$ or $c_i = -1$ when $c'_i < -1$. Below is a pseudo code for such process.

1. Get guess c_i to move
2. Sample real number Δc_i from the normal distribution whoose mean and standard derivation is 0 and step↩ _size, respectively.
3. add such real number to guess $c'_i = c_i + \Delta c_i$
4. If $c'_i > 1$, then set $c'_i = 1$.
5. If $c'_i < -1$, then set $c'_i = -1$.
6. Return c'_i.

Chapter 5

Monte Carlo Metropolis Method with Kepler action

5.1 Kepler Action

Define path $p(t) = (\zeta(t), \theta(t))$ then the Kepler Lagrangian $L(t, p(t), p'(t))$ is given by

$$L(t, p(t), p'(t)) = \frac{1}{2}\left(\zeta'^2(t) + \zeta^2(t)\theta'^2(t)\right) + \frac{1}{\zeta(t)} \tag{5.1}$$

With boundary condition $p(t_0) = p_0$ and $p(t_f) = p_f$, Kepler path (the solution of Kepler problem) minimizes following action

$$S = \int_{t_0}^{t_f} L(t, p(t), p'(t)) \tag{5.2}$$

If we set $p(t_0) = (\zeta_{min}, 0)$ and $p(t_f) = (\zeta_{max}, pi)$, where ζ_{min} is periapsis and ζ_{max} is apoapsis. Then automately ζ_{max} and t_f are determined as following equations.

$$\zeta_{max} = \frac{\zeta_{min}}{2\zeta_{min} - 1} \tag{5.3}$$

$$t_f = t_0 + \pi \left(\frac{\zeta_{min} + \zeta_{max}}{2}\right)^{3/2} \tag{5.4}$$

However the Kepler lagrangian (5.1) has singularities when $\zeta(t) = 0$ for some $t \in (t_0, t_f)$. Such singularities make Kepler action integral (5.2) may diverge. We know that for Kepler path $\zeta_{min} \le \zeta(t) \le \zeta_{max}$ and by the relation of ζ_{min} and ζ_{max} (5.3) , ζ_{min} should be $0.5 < \zeta_{min} < 1$. Thus, I define modified Kepler Lagrangian $\hat{L}(t, p(t), p'(t))$ as following.

$$\hat{L}(t, p(t), p'(t)) = \begin{cases} 100 + \frac{1}{2}\left(\zeta'^2(t) + \zeta^2(t)\theta'^2(t)\right) & \text{,if } \zeta(t) < 0.01 \\ L(t, p(t), p'(t)) & \text{otherwise} \end{cases} \tag{5.5}$$

Then modified Kepler action $\hat{S}$ is also defined by

$$\hat{S} = \int_{t_0}^{t_f} \hat{L}(t, p(t), p'(t)) \tag{5.6}$$

For good path (i.e. $\zeta_{min} \le \zeta(t) \le \zeta_{max}$), $\hat{S}_{good} = S_{good}$ and for bad path (i.e. for some t, $\zeta(t) < 0.01$) $\hat{S}_{bad} \gg \hat{S}_{good}$. Hence, Kepler path also minimizes modified Kepler action $\hat{S}$. Therefore, during Monte Carlo Metropolis optimization process, I use modified Kepler action instead of Kepler action to avoid singularity of action integral at sampled path.

5.2 Initial condition

I set initial time $t_0 = 0$ and periapsis $\zeta_{min} = 0.9$.

5.3 Optimization Setup

I use following setup during optimization.

- Period of fourier function T: $2t_f$.
- Single precision
  - setup: 1
    1. Relative tolerance of action integration: 10^{-4}
    2. step_size: 0.01
    3. λ: 100
    4. Number of iteration: 10^6
  - setup: 2
    1. Relative tolerance of action integration: 10^{-4}
    2. step_size: 0.001
    3. λ: 10000
    4. Number of iteration: 10^6
- Double precision
  - setup: 1
    1. Relative tolerance of action integration: 10^{-8}
    2. step_size: 0.01
    3. λ: 100
    4. Number of iteration: 10^6
  - setup: 2
    1. Relative tolerance of action integration: 10^{-8}
    2. step_size: 0.001
    3. λ: 10000
    4. Number of iteration: 10^6

5.4 Trajectory

Trajectory of path can be evaluated by following relation

$$\begin{aligned} x(t) &= \zeta(t)\cos\theta(t) \\ y(t) &= \zeta(t)\sin\theta(t) \end{aligned} \tag{5.7}$$

5.5 Error Estimation

Kepler problem can be solved via Newton's equation. Reference Kepler path is calculated by solving Newton's equation via finite difference method with 10^5 steps and double precision level. Difference of optimized path and reference Kepler path can be measured by following two ways.

1. Difference of action
   $$\Delta S = S_{min} - S_{exact} \tag{5.8}$$
   By the minimum action principle $\Delta S > 0$.

2. Relative L_2 error

   L_2 norm of path defined as following

$$L_2 = \left(\int_{t_0}^{t_f} x^2(t) + y^2(t) \mathrm{d}t \right)^{1/2} \tag{5.9}$$

   So, relative L_2 error can be estimated by

$$\mathrm{rel}_{L_2} = \frac{1}{L_{2_{exact}}} \left(\int_{t_0}^{t_f} (x(t) - x_{exact}(t))^2 + (y(t) - y_{exact}(t))^2 \mathrm{d}t \right)^{1/2} \tag{5.10}$$

Chapter 6

Result: Fourier function

Monte Carlo Metropolis Optimization Results

n_f	precision	setup	acceptance ratio	S_{min}
1	single	1	0.874679	5.03173
2	single	1	0.727210	4.75809
3	single	1	0.574712	4.74397
3	single	2	0.591197	4.74250
4	single	2	0.419363	4.74191
1	double	1	0.870009	5.03173
2	double	1	0.727406	4.75818
3	double	1	0.561235	4.74380
3	double	2	0.588501	4.74251
4	double	2	0.427474	4.74184

Above table summarizes optimization results with various setup and precision.

6.1 Optimized Path

Optimized path with various n_f, setup and precision are plotted below.

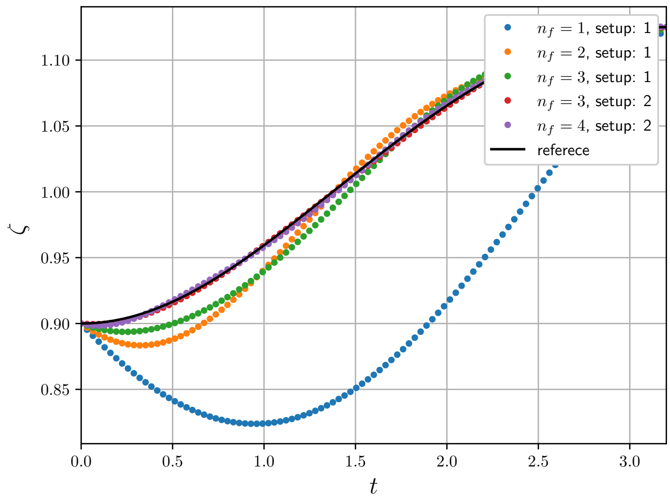


Figure 6.1 Zeta: single precision

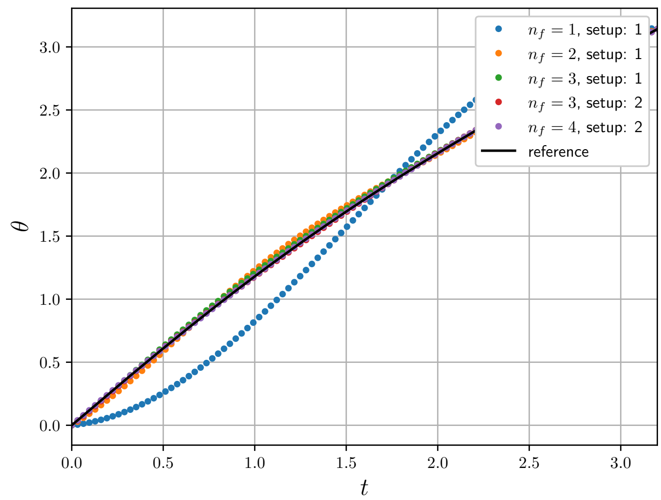


Figure 6.2 Theta: single precision

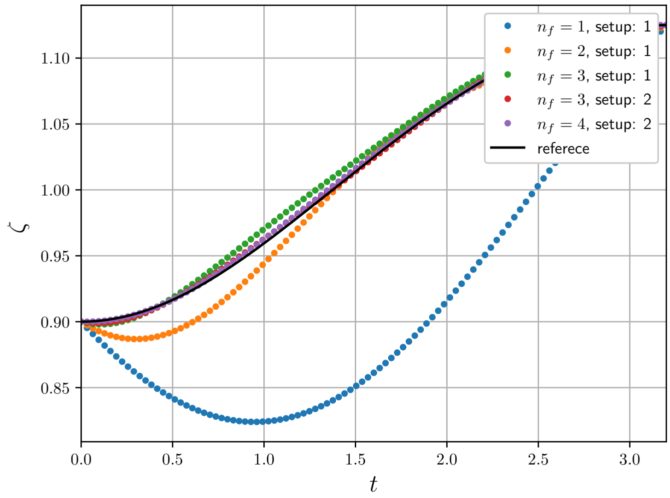


Figure 6.3 Zeta: double precision

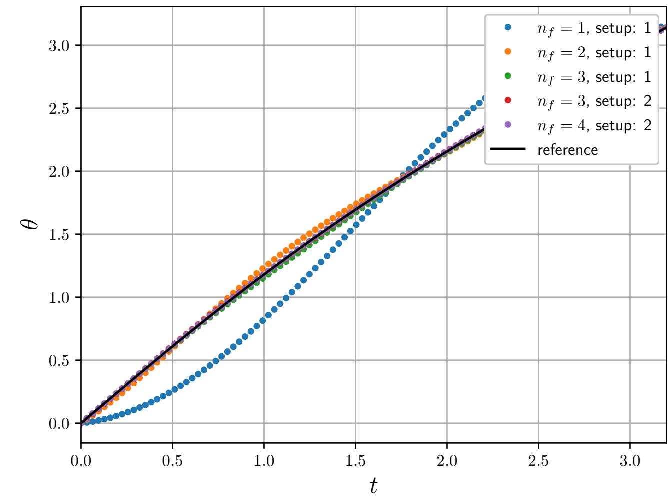


Figure 6.4 Theta: double precision

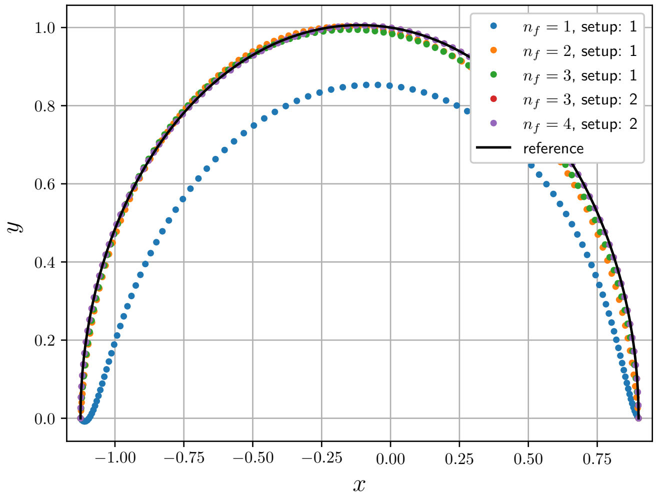


Figure 6.5 Trajectory: single precision

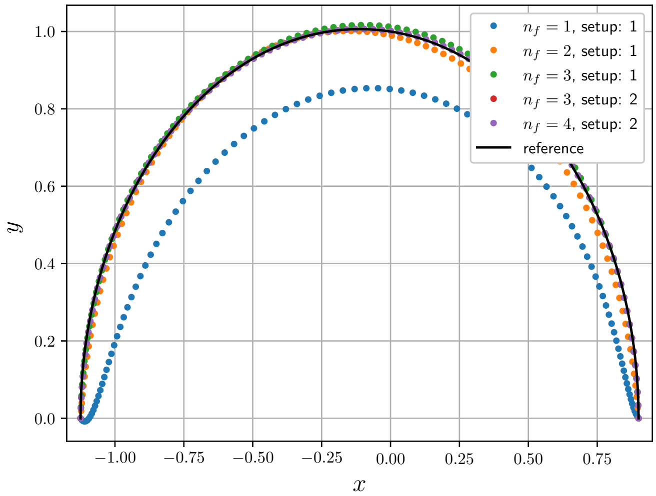


Figure 6.6 Trajectory: double precision

Both single and double precision, the optimized fourier path reaches to exact path as n_f increases. At $n_f = 3$, optimized path is improved as step_size decreases from 0.01 to 0.001 and λ increases from 10^2 to 10^4 (compare $n_f = 3$, setup:1 and $n_f = 3$, setup: 2). To see how setup_size and λ affect optimization process we need to monitor optimization process.

6.2 Monitoring Optimization process

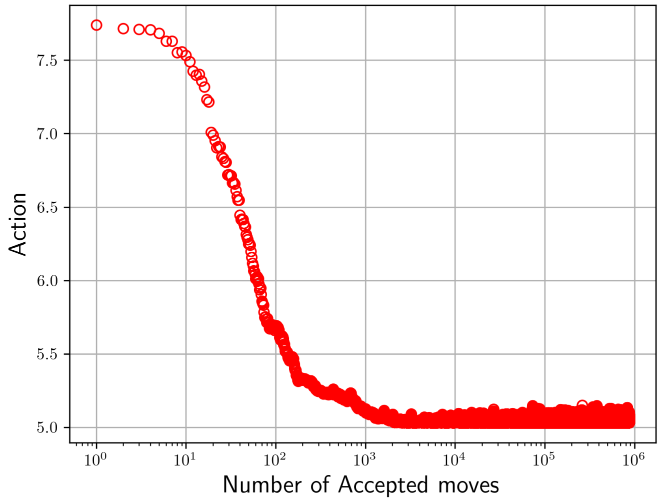


Figure 6.7 Monitor: n_f=1, setup: 1, single precision

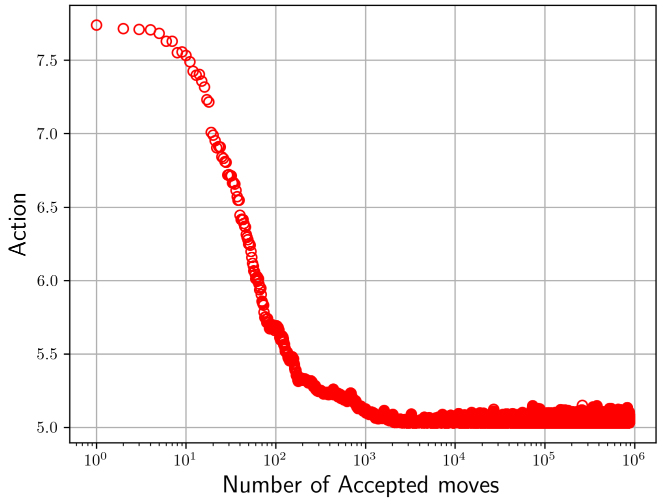


Figure 6.8 Monitor: n_f=2, setup: 1, single precision

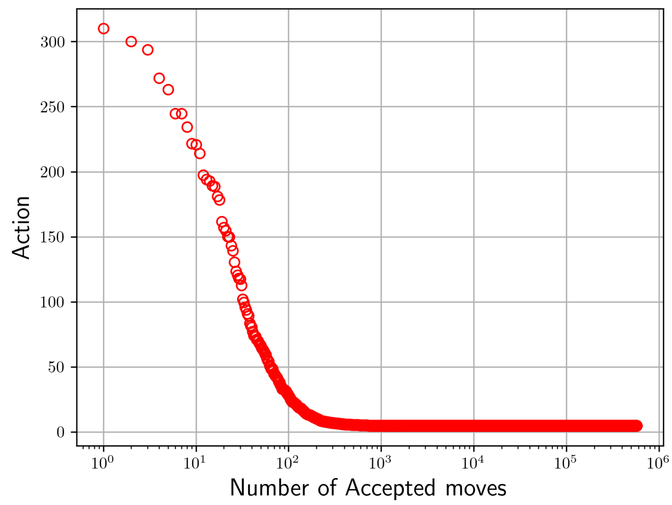


Figure 6.9 Monitor: n_f=3, setup: 1, single precision

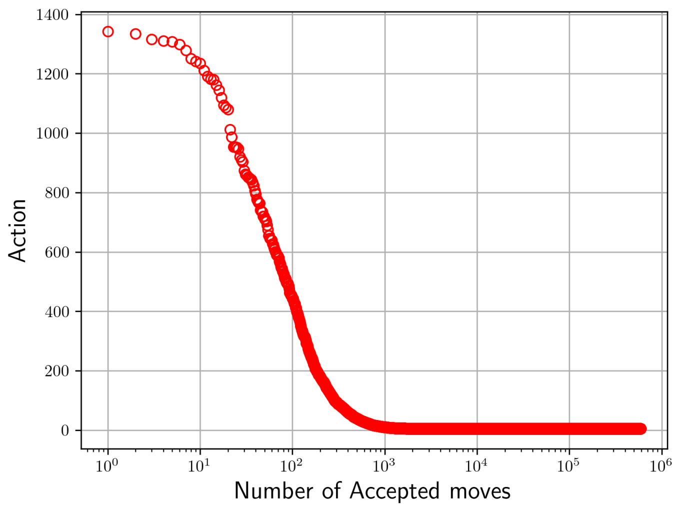


Figure 6.10 Monitor: n_f=3, setup: 2, single precision

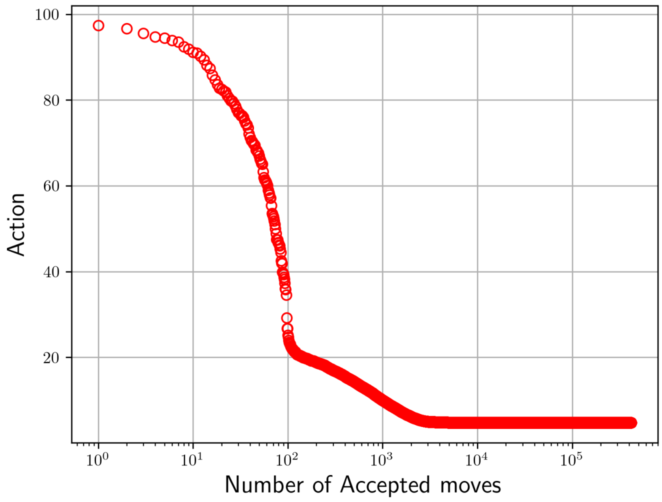


Figure 6.11 Monitor: n_f=4, setup: 2, single precision

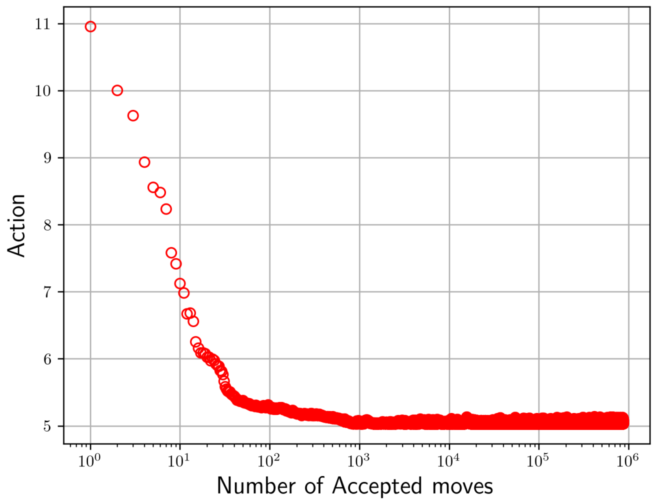


Figure 6.12 Monitor: n_f=1, setup: 1, double precision

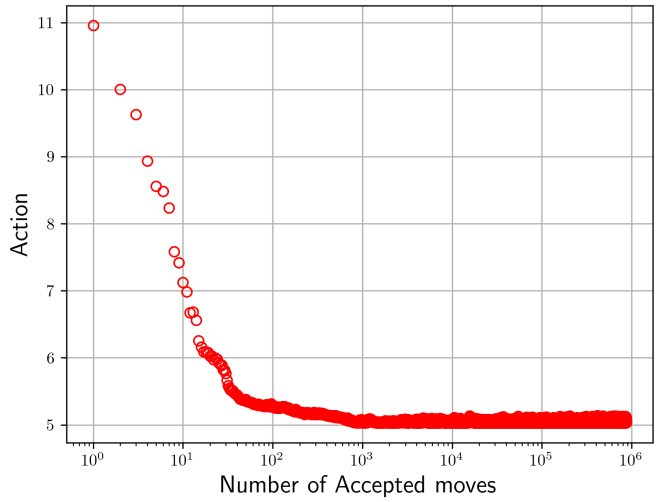


Figure 6.13 Monitor: n_f=2, setup: 1, double precision

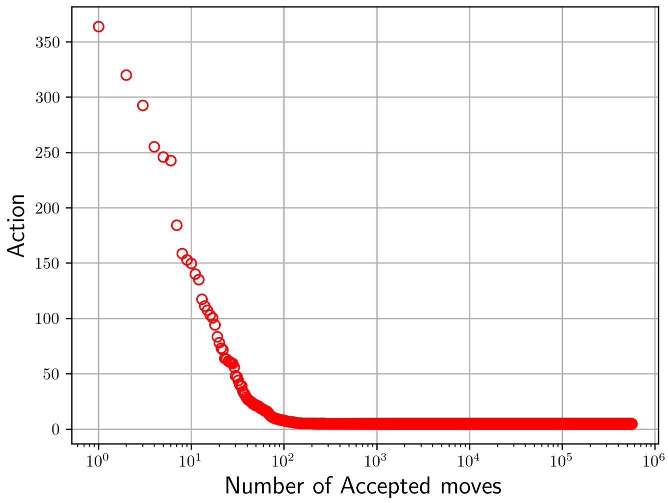


Figure 6.14 Monitor: n_f=3, setup: 1, double precision

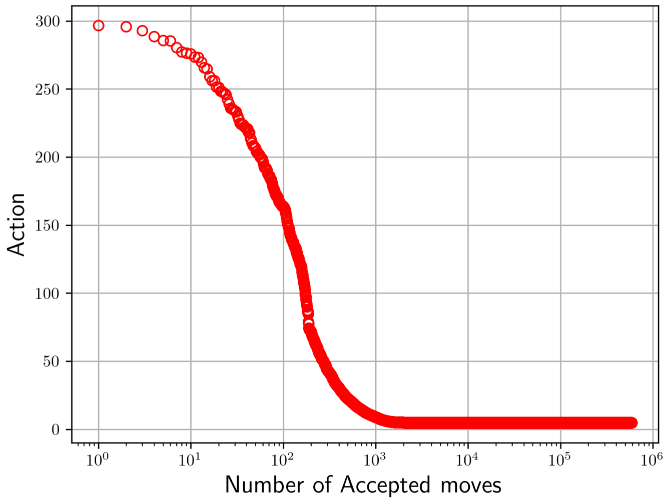


Figure 6.15 Monitor: n_f=3, setup: 2, double precision

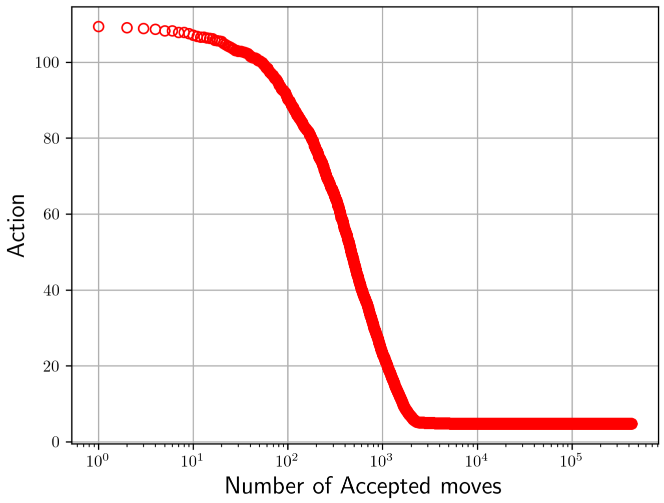


Figure 6.16 Monitor: n_f=4, setup: 2, double precision

Markov Chain converges at $n > 10^4$. Now extract distribution from the Markov Chains.

6.3 Converged distribution

To extract converged distribution from the Markov Chain, first we need to discard unconverged Markov Chain. To do this I burns first 10^4 chains. Now make histrograms using converged Markov Chain. Then we could estimate converged distribution as following plots.

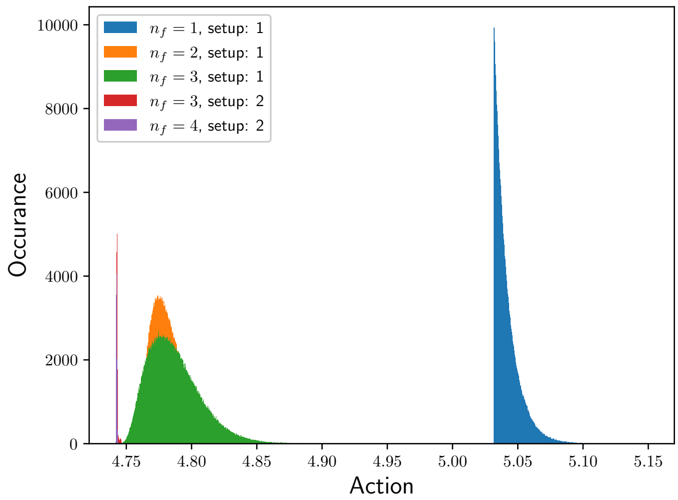


Figure 6.17 Distribution: single precision

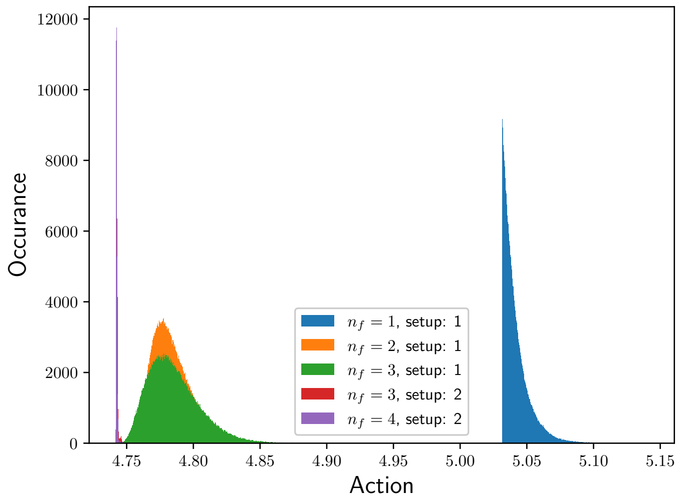


Figure 6.18 Distribution: double precision

Distribution: setup 1 table

n_f	**precision**	S_{min}	S_{most}	$S_{most} - S_{min}$	$\mathrm{prob}_{min}/\mathrm{prob}_{most}$
1	single	5.03173	5.03179	0.00006	1.00000
2	single	4.75809	4.77398	0.01589	0.00226
3	single	4.74397	4.77475	0.03078	0.00037
1	double	5.03173	5.03179	0.00006	1.00000
2	double	4.75818	4.77783	0.01965	0.00226
3	double	4.74380	4.77757	0.03377	0.00039

At $n_f = 1$, degeneracy factor $P(S)$ nearly constant, so most probable probable path is same as least action path. However since $n_f = 1$ is not enough to describe exact path, minimum action is far greater than exact action

$S_{exact} = 4.74175$. At $n_f = 2$, both minimum action and most probable action are far lower than minimum action of $n_f = 1$. However due to the degeneracy factor $P(S)$, most probable action and least probable action are different. At $n_f = 3$, due to the degeneracy factor, most probable action is almost same as the one of $n_f = 2$. Contrary to this, minimum action of $n_f = 3$ much less than that of $n_f = 2$. Therefore, we should not only increase n_f but also λ to obtain the minimum action path which is more close to the exact least action one.

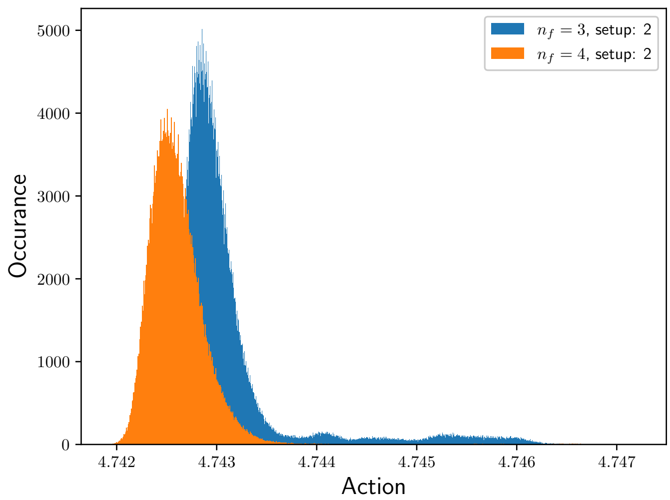


Figure 6.19 Distribution: setup 2, single precision

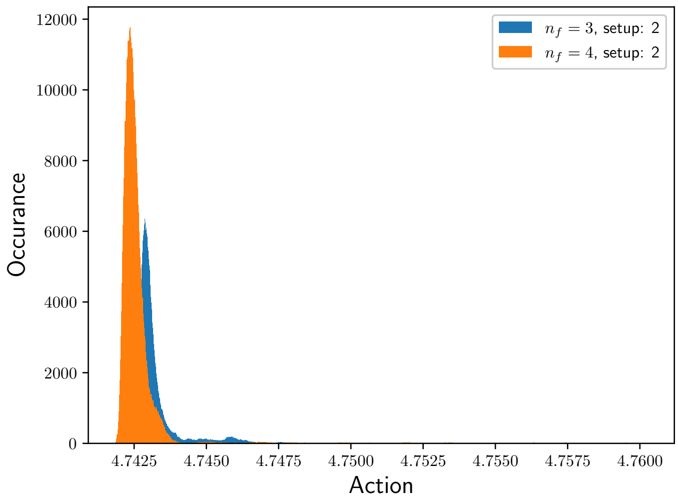


Figure 6.20 Distribution: setup 2, double precision

Distribution: setup 2 table

n_f	**precision**	S_{min}	S_{most}	$S_{most} - S_{min}$	$\text{prob}_{min}/\text{prob}_{most}$
3	single	4.74250	4.74285	0.00035	0.00080
4	single	4.74191	4.74251	0.00060	0.00025
3	double	4.74251	4.74288	0.00037	0.00016
4	double	4.74184	4.74236	0.00052	0.00068

Since our target distribution is

$$
\begin{aligned}
g(S) &= P(S)\exp\left(-\lambda(S - S_{min})\right) \\
&= P(S)\exp\left(-\lambda/2(S - S_{min})\right)\exp\left(-\lambda/2(S - S_{min})\right)
\end{aligned} \tag{6.1}
$$

, contribution of degeneracy factor is lowered as λ increases. This decreases the value of most probable action and increases the likelihood of finding much smaller value of action. So, as parameter λ increases from 10^2 to 10^4, both most probable and minimum action decrease. Also, the difference between most probable action and minimum action is lowered. (Compare setp 1 table $n_f = 3$, $n_f = 4$.) At $n_f = 4$ with $\lambda = 10^4$, relative difference of the minimum action and exact action is smaller than 0.01%, so I stop optimization at $n_f = 4$.

6.4 Error Analysis

Below table summerizes error analysis of the optimization process.

Error Analysis

n_f	precision	setup	S_{min}	S_{exact}	$\Delta S = S_{min} - S_{exact}$	relative L_2 error
1	single	1	5.03173	4.74175	0.28998	0.13660
2	single	1	4.75809		0.01634	0.01777
3	single	1	4.74397		0.00222	0.00988
3	single	2	4.74250		0.00075	0.00198
4	single	2	4.74191		0.00016	0.00144
1	double	1	5.03173		0.28998	0.13659
2	double	1	4.75818		0.01643	0.01749
3	double	1	4.74380		0.00205	0.00924
3	double	2	4.74251		0.00076	0.00215
4	double	2	4.74184		0.00009	0.00101

As n_f increases and parameter λ increase, minimum action S_{min} converges rapidly to exact one. However, relative L_2 error decreases slower than convergence of S_{min}, since we minimize action S, rather than relative L_2 error. Although of this, at $n_f = 3, 4$ with parameter $\lambda = 10^4$, relative L_2 error is much small (less than 0.5%.).

Chapter 7

Result: Bezier Curve

How number of basis function and parameter λ affect optimization is summarized in Result: Fourier function. In Result: Fourier function, six basis function ($n_f = 3$) with optimization parameter $\lambda = 10^4$ is enough to find minimal path closed to exact one. So, I use six order bezier curve to approximate path and set optimization parameter $\lambda = 10^4$ to suppress degeneracy factor. Below table shows optimization result with various precision.

Monte Carlo Metropolis Optimization Results

n	precision	setup	acceptance ratio	S_{min}
6	single	2	0.713119	4.74177
6	double	2	0.729769	4.74177

7.1 Optimized Path

Optimized path with various precision are plotted below.

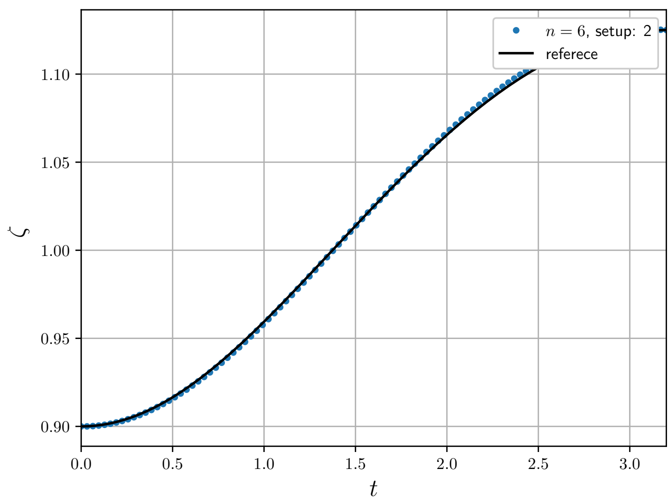


Figure 7.1 Zeta: single precision

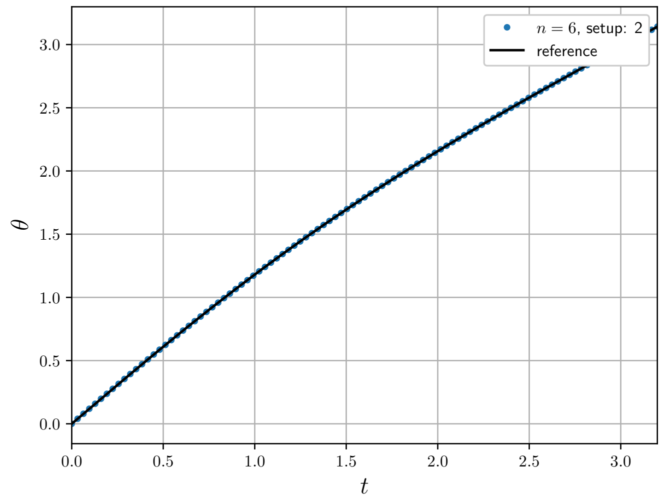


Figure 7.2 Theta: single precision

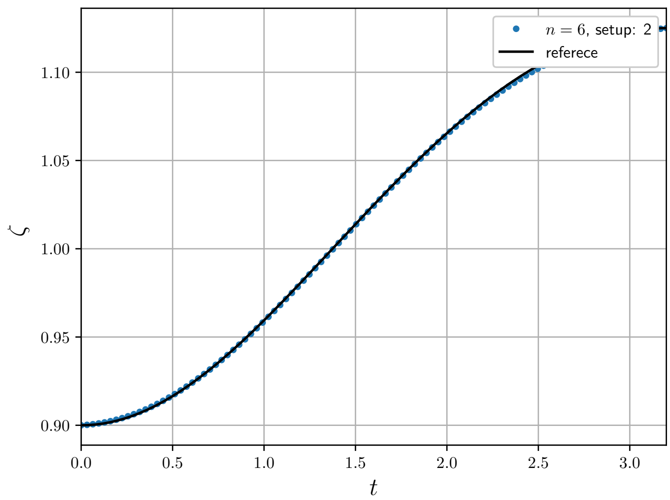


Figure 7.3 Zeta: double precision

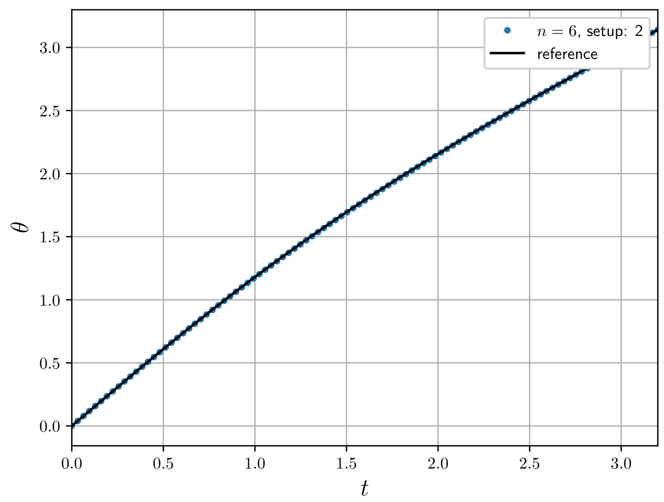


Figure 7.4 Theta: double precision

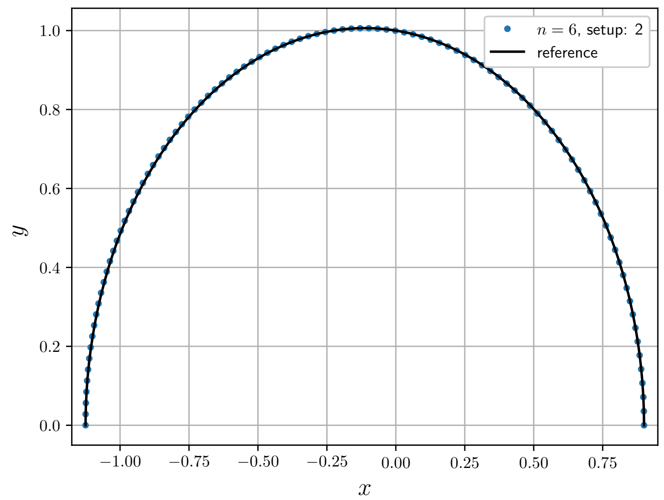


Figure 7.5 Trajectory: single precision

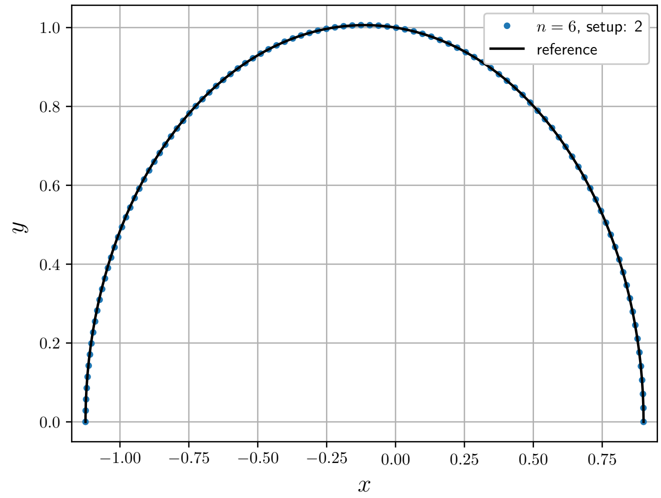


Figure 7.6 Trajectory: double precision

As you can see the minimum action path which approximated by six order bezier curve is almost same as the exact one.

7.2 Monitoring Optimization process

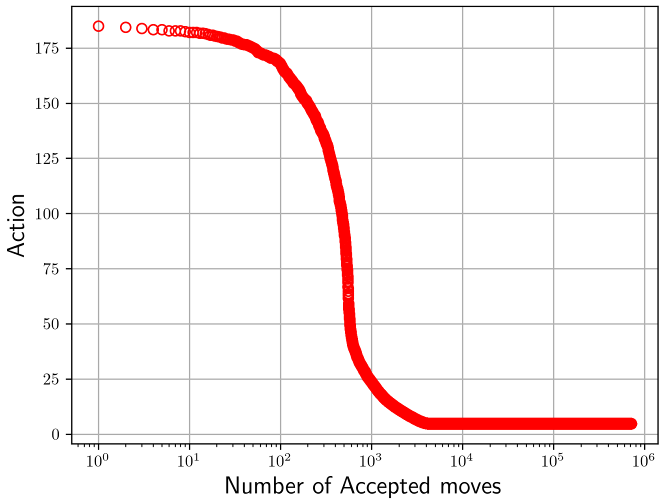


Figure 7.7 Monitor: n=6, setup: 2, single precision

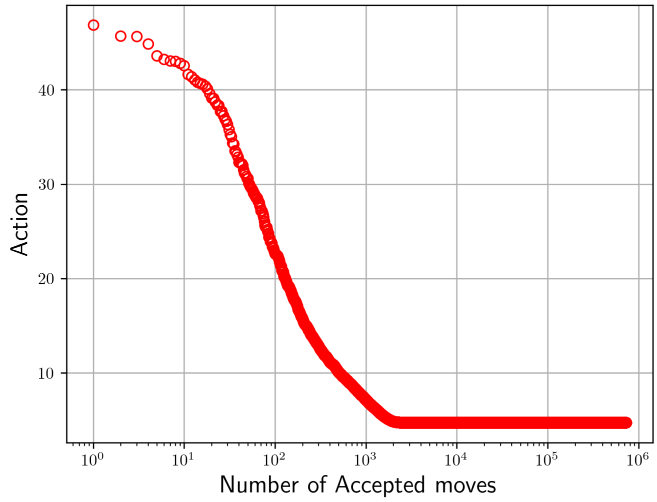


Figure 7.8 Monitor: n=6, setup: 2, single precision

Both single and double precision, Markov Chains are converged at $n = 10^4$.

7.3 Converged distribution

To get converged distribution, I burns first 10^4 Markov Chains.

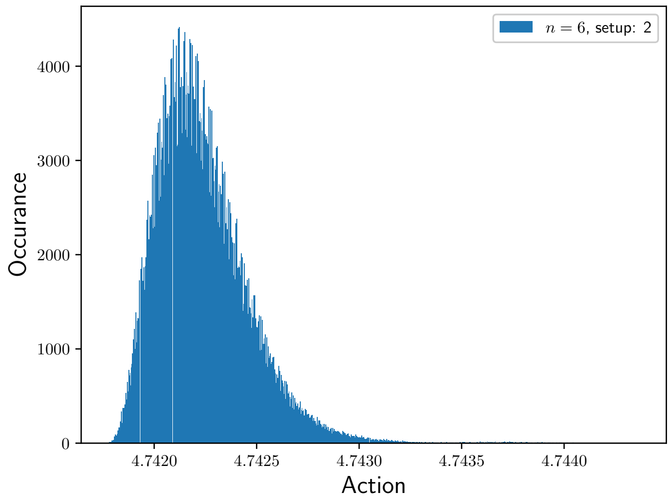


Figure 7.9 Distribution: single precision

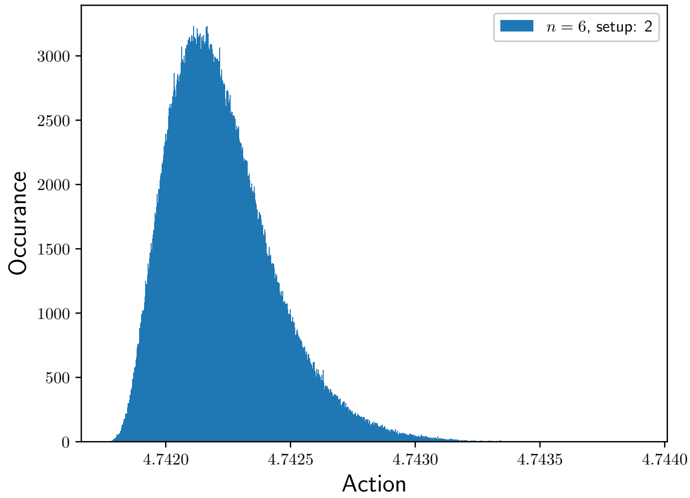


Figure 7.10 Distribution: double precision

Distribution

n	precision	S_{min}	S_{most}	$S_{most} - S_{min}$	$\mathrm{prob}_{min}/\mathrm{prob}_{most}$
6	single	4.74177	4.74212	0.00035	0.00091
6	double	4.74177	4.74217	0.00040	0.00031

Similar to fourier function result Result: Fourier function Converged distribution, due to the degeneracy factor $P(S)$, least action path is not same as most probable path. Although of this, minimum action is very close to the action of exact one. Therefore we can conclude that $\lambda = 10^4$ is enough to lower the contribution of degeneracy factor.

7.4 Error Analysis

Below table summerizes error analysis of the optimization process.

Error Analysis

n	precision	setup	S_{min}	S_{exact}	$\Delta S = S_{min} - S_{exact}$	relative L_2 error
6	single	2	4.74177	4.74175	0.00002	0.00064
6	double	2	4.74177		0.00002	0.00168

Similar to fourier function result Result: Fourier function Error Analysis, relative difference between minimum action and exact one much much smaller than relative L_2 error. (0.0004% vs. 0.2%.)

Chapter 8

Conclusion

In conclusion, Markov Chain Monte Carlo based minimization of action method described in `Entropy 2020, 22(9), 916` is reliable (relative error of minimum action is smaller than 0.001%) and fast (less than 10^6 iteration need to find exact solution) method to search exact minimum action path with boundary condition. However, due to the degeneracy factor in target distribution, we should varying not only number or order of basis function but also acceptance parameter λ.

Chapter 9

Module Index

9.1 Modules

Here is a list of all modules:

Chapter 10

Hierarchical Index

10.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 11

Class Index

11.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 12

File Index

12.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 13

Module Documentation

13.1 libpath

template library which

Files

- file action.hpp

    header file for evaluation of the action
- file bezier.hpp

    headerfile for bezier curve and path approximated by bezier curve
- file bezier_path.hpp

    headerfile for path approximated by bezier curve
- file fourier.hpp

    headerfile for fourier function
- file fourier_path.hpp

    headerfile for path approximated by fourier function
- file node_weight_table.hpp

    table for node and weights

Classes

- class libpath::action< T, Path, Lag >

    class which computes action
- class libpath::bezier< T >

    Class which defines n th order Bezier curve.
- class libpath::bezier_path< T >

    Class which approximate path by n th order Bezier curve.
- class libpath::fourier< T >

    Class which defines fourier function.
- class libpath::fourier_path< T >

    Class for the path approximated by fourier function.
- class libpath::gau_kron_table< T, N >

    table for gauss kronrod node and weights

13.1.1 Detailed Description

template library which

1. define basis function (fourier function, bezier curve)
2. approximate path using support function
3. compute the action of path approximated by basis function

13.2 libmcm

template library which minimize action via Monte Carlo Metropolis Method

Files

- file mcm.hpp

  headerfile for the minimization of the action by Monte Carlo Metropolis method described in `Entropy 2020, 22(9), 916`
- file mcm_bezier.hpp

  headerfile for mcm_bezier class which is derivated by libmcm::mcm class
- file mcm_fourier.hpp

  headerfile for mcm_fourier class which is derivated by libmcm::mcm class

Classes

- class libmcm::mcm$<$ T, Basis, Path, Lag $>$

  class to Minimize the action via Monte Carlo Metropolis Method. It uses mt19937 random number generator to generates distribution. Moreover it samples path via random walk. To make random walk, it samples real number from normal distribution and move path by the sampled real number.

13.2.1 Detailed Description

template library which minimize action via Monte Carlo Metropolis Method

Chapter 14

Class Documentation

14.1 libpath::action< T, Path, Lag > Class Template Reference

class which computes action

`#include <action.hpp>`

Public Member Functions

- action ()
  initialize action class
- action (std::vector< Path > path)
  initialize action class
- action (T rel_tol)
  initialize action class
- action (T rel_tol, std::vector< Path > path)
  initialize action class
- action (const action< T, Path, Lag > &copy)
  copy constructer of action class
- action< T, Path, Lag > & operator= (const action< T, Path, Lag > &copy)
  overloading of assignment operator for action class
- void update (std::vector< Path > path)
  update path
- void update (T rel_tol)
  update relative tolerance
- void update (std::vector< Path > path, T rel_tol)
  update relative tolerance and path
- bool is_vaild ()
  check vaildity of path
- T eval (T &e)
  evaluate the action of given path by default method: (G15, K31) Gauss–Kronrod quadrature method
- T eval (int method, unsigned int n, T &e)
  evaluate the action of given path

14.1.1 Detailed Description

template<typename T, typename Path, typename Lag>
class libpath::action< T, Path, Lag >

class which computes action

Parameters

T	precision should be one of float, double and long double
Path	type of path should be one of fourier_path or bezier_path
Lag	lagrangian of action functor class which has time, path and derivative of path as variable and it returns value of lagrangian at given time

Note

Define L_1 norm of lagrangian as

$$L_1 = \int_{t_0}^{t_f} |L(t, p(t), p'(t))| \, \mathrm{d}t \tag{14.1}$$

Then the numerical integration is converged when estimated error e is

$$e < r \cdot L_1 \tag{14.2}$$

, where r is relative tolerance of action integral.

Warning

If you give invaild path, then eval method will return always zero.

For gauss-kronrod quadrature method,

See also

`Math. Comp. 22 (1968), 847-856.`
`Commun. ACM 16, 11 (Nov. 1973), 694-699.`
`ACM Comput. Surv. 44, 4, Article 22 (August 2012)`

For tanh-sinh quadrature method,

See also

`Publ. RIMS, Kyoto Univ. 9 (1974), 721-741`
`Publ. RIMS, Kyoto Univ. 41 (2005), 897-935`
`David H. Bailey, Tanh-Sinh High-Precision Quadrature`

14.1.2 Constructor & Destructor Documentation

14.1.2.1 action() [1/3]

```
template<typename T , typename Path , typename Lag >
libpath::action< T, Path, Lag >::action (
            std::vector< Path > path ) [inline]
```

initialize action class

Parameters

path	path

14.1.2.2 action() [2/3]

```
template<typename T , typename Path , typename Lag >
libpath::action< T, Path, Lag >::action (
            T rel_tol ) [inline]
```

initialize action class

Parameters

rel_tol	relative tolerance

14.1.2.3 action() [3/3]

```
template<typename T , typename Path , typename Lag >
libpath::action< T, Path, Lag >::action (
            T rel_tol,
            std::vector< Path > path ) [inline]
```

initialize action class

Parameters

rel_tol	absoulte tolerance
path	path

14.1.3 Member Function Documentation

14.1.3.1 eval() [1/2]

```
template<typename T , typename Path , typename Lag >
T libpath::action< T, Path, Lag >::eval (
            int method,
            unsigned int n,
            T & e )
```

evaluate the action of given path

Parameters

	method	numerical integration method • method 0: Gauss-Kronrod quadrature method • method 1: Tanh-Sinh quadrature method
	n	additional parameter • order of Gauss-Kronrod quadrature if method equals to 0, currently, only support n=15, 21, 31, 41, 51, 61. • maximum depth if method equals to 1.
out	*e*	estimated absolute error

Returns

action of given path

14.1.3.2 eval() [2/2]

```
template<typename T , typename Path , typename Lag >
T libpath::action< T, Path, Lag >::eval (
            T & e )
```

evaluate the action of given path by default method: (G15, K31) Gauss–Kronrod quadrature method

Parameters

out	*e*	estimated absolute error

Returns

action of given path

14.1.3.3 is_vaild()

```
template<typename T , typename Path , typename Lag >
bool libpath::action< T, Path, Lag >::is_vaild ( ) [inline]
```

check vaildity of path

Returns

vaildity of path

14.1.3.4 update() [1/3]

```
template<typename T , typename Path , typename Lag >
void libpath::action< T, Path, Lag >::update (
            std::vector< Path > path ) [inline]
```

update path

Parameters

path	path to update

14.1.3.5 update() [2/3]

```
template<typename T , typename Path , typename Lag >
void libpath::action< T, Path, Lag >::update (
            std::vector< Path > path,
            T rel_tol ) [inline]
```

update relative tolerance and path

Parameters

path	path to update
rel_tol	relative tolerance

14.1.3.6 update() [3/3]

```
template<typename T , typename Path , typename Lag >
void libpath::action< T, Path, Lag >::update (
            T rel_tol ) [inline]
```

update relative tolerance

Parameters

rel_tol	relative tolerance

The documentation for this class was generated from the following file:

- libpath/action.hpp

14.2 libpath::bezier< T > Class Template Reference

Class which defines n th order Bezier curve.

```
#include <bezier.hpp>
```

Public Member Functions

- bezier ()
  initialize bezier class
- bezier (unsigned int n_, std::vector< T > c_)
  initialize bezier class
- bezier (unsigned int n_, T dummy, std::vector< T > c_)
  initialize bezier class
- bezier (const bezier< T > &copy)
  copy constructor of bezier class
- bezier< T > & operator= (const bezier< T > &copy)
  overloading of assignment operator for bezier class
- unsigned int terms (unsigned int n_) const
  Inform how many control points need to define bezier curve.
- T get_first () const
  get first control point
- T get_last () const
  get last control point
- std::vector< T > get_crtl_pts () const
  get control points of bezier curve
- void update (std::vector< T > c_)
  update control points
- void update_first (T c_0)
  update first control point
- void update_last (T c_n)
  update last control point
- T eval (T t)
  evaluate bezier curve at given point $0 \leq t \leq 1$.
- std::vector< T > eval (std::vector< T > t)
  evaluate bezier curve at given points $0 \leq t \leq 1$.
- T deriv (T t)
  evaluate derivative of bezier curve at given point $0 \leq t \leq 1$.
- std::vector< T > deriv (std::vector< T > t)
  evaluate bezier curve at given points $0 \leq t \leq 1$.

14.2.1 Detailed Description

template<typename T>
class libpath::bezier< T >

Class which defines n th order Bezier curve.

$$B(t) = \sum_{i=0}^{n} c_i \binom{n}{i} (1-t)^{n-i} t^i \tag{14.3}$$

,where c_i is control points.

Parameters

T	precision should be one of float, double or long double.

Note

For n th order Bezier curve, you should give $n + 1$ control points.

14.2.2 Constructor & Destructor Documentation

14.2.2.1 bezier() [1/2]

```
template<typename T >
libpath::bezier< T >::bezier (
            unsigned int n_,
            std::vector< T > c_ )  [inline]
```

initialize bezier class

Parameters

*n_*	order of Bezier curve
*c_*	control points of Bezier curve

14.2.2.2 bezier() [2/2]

```
template<typename T >
libpath::bezier< T >::bezier (
            unsigned int n_,
            T dummy,
            std::vector< T > c_ )  [inline]
```

initialize bezier class

Parameters

*n_*	order of Bezier curve
dummy	dummy parameter used for consistence
*c_*	control points of Bezier curve

14.2.3 Member Function Documentation

14.2.3.1 deriv() [1/2]

```
template<typename T >
std::vector<T> libpath::bezier< T >::deriv (
            std::vector< T > t )
```

evaluate bezier curve at given points $0 \leq t \leq 1$.

Parameters

t	points at which bezier curve is evaluated

14.2.3.2 deriv() [2/2]

```
template<typename T >
T libpath::bezier< T >::deriv (
            T t )
```

evaluate derivative of bezier curve at given point $0 \leq t \leq 1$.

Parameters

t	point at which bezier curve is evaluated

14.2.3.3 eval() [1/2]

```
template<typename T >
std::vector<T> libpath::bezier< T >::eval (
            std::vector< T > t )
```

evaluate bezier curve at given points $0 \leq t \leq 1$.

Parameters

t	points at which bezier curve is evaluated

14.2.3.4 eval() [2/2]

```
template<typename T >
T libpath::bezier< T >::eval (
            T t )
```

evaluate bezier curve at given point $0 \leq t \leq 1$.

Parameters

t	point at which bezier curve is evaluated

14.2.3.5 get_crtl_pts()

```
template<typename T >
std::vector<T> libpath::bezier< T >::get_crtl_pts ( ) const [inline]
```

get control points of bezier curve

Returns

control points of bezier curve

14.2.3.6 get_first()

```
template<typename T >
T libpath::bezier< T >::get_first ( ) const [inline]
```

get first control point

Returns

first control point

14.2.3.7 get_last()

```
template<typename T >
T libpath::bezier< T >::get_last ( ) const [inline]
```

get last control point

Returns

last control point

14.2.3.8 terms()

```
template<typename T >
unsigned int libpath::bezier< T >::terms (
            unsigned int n_ ) const [inline]
```

Inform how many control points need to define bezier curve.

Parameters

n↩ _↩	order of Bezier

Returns

number of terms need to define n th order bezier curve

14.2.3.9 update()

```
template<typename T >
void libpath::bezier< T >::update (
            std::vector< T > c_ )  [inline]
```

update control points

Parameters

c↩ _↩	control points to update

14.2.3.10 update_first()

```
template<typename T >
void libpath::bezier< T >::update_first (
            T c_0 )  [inline]
```

update first control point

Parameters

c↩ *_0*	first control point to update

14.2.3.11 update_last()

```
template<typename T >
void libpath::bezier< T >::update_last (
            T c_n )  [inline]
```

update last control point

Parameters

c_n	last control point to update

The documentation for this class was generated from the following file:

- libpath/bezier.hpp

14.3 libpath::bezier_path< T > Class Template Reference

Class which approximate path by n th order Bezier curve.

`#include <bezier_path.hpp>`

Public Member Functions

- bezier_path ()
  initialize bezier_path class
- bezier_path (T t_init_, T t_final_, T p_init_, T p_final_)
  initialize bezier_path class
- bezier_path (T t_init_, T t_final_, T p_init_, T p_final_, bezier< T > B_)
  initialize bezier_path class
- bezier_path (const bezier_path< T > &copy)
  copy constructor of bezier_path class
- bezier_path< T > & operator= (const bezier_path< T > &copy)
  overloading of assignment operator for bezier_path class
- void update (bezier< T > B_)
  update bezier curve
- bool is_vaild () const
  check whether or not the bezier curve is valid to approximate path
- std::tuple< T, T > get_endtimes () const
  get initial and final time of path
- std::vector< T > get_ctrl_pts () const
  get modified control points
- T get_scaler () const
  get the second scaler parameter
- T eval (T t)
  evaluate path approximated by bezier curve at given point
- std::vector< T > eval (std::vector< T > t)
  evaluate path approximated by bezier curve at given points
- T deriv (T t)
  evaluate derivative of bezier_path at given point
- std::vector< T > deriv (std::vector< T > t)
  evaluate derivative of bezier_path at given points

14.3.1 Detailed Description

template$<$typename T$>$
class libpath::bezier_path$<$ T $>$

Class which approximate path by n th order Bezier curve.

$$\Phi(t) = \text{scale}_2 \cdot B\left(\frac{t - t_i}{t_f - t_i}, n, \{c_0, c_1, \dots, \text{scale}_1 \cdot c_n\}\right) \tag{14.4}$$

,where c_i is control points of bezier curve, scale_1 is the first scaling parameter to match ratio of initial and final value of path and scale_2 is the second scaling parameter to match boundary condition.

Parameters

T	precision should be one of float, double or long double.

Note

If initial value of path, given by boundary condition, is zero then it sets $c_0 = 0$.

Warning

If bezier curve is not appropriate to approximate path then eval and deriv method returns always zero.

14.3.2 Constructor & Destructor Documentation

14.3.2.1 bezier_path() [1/2]

```
template<typename T >
libpath::bezier_path< T >::bezier_path (
            T t_init_,
            T t_final_,
            T p_init_,
            T p_final_ )  [inline]
```

initialize bezier_path class

Parameters

*t_init_*	initial time
*t_final_*	final time
*p_init_*	initial value of path
*p_final_*	final value of path

14.3.2.2 bezier_path() [2/2]

```
template<typename T >
libpath::bezier_path< T >::bezier_path (
            T t_init_,
            T t_final_,
            T p_init_,
            T p_final_,
            bezier< T > B_ )  [inline]
```

initialize bezier_path class

Parameters

*t_init_*	initial time
*t_final_*	final time
*p_init_*	initial value of path
*p_final_*	final value of path
*B_*	Bezier curve used to approximate path

14.3.3 Member Function Documentation

14.3.3.1 deriv() [1/2]

```
template<typename T >
std::vector<T> libpath::bezier_path< T >::deriv (
            std::vector< T > t )
```

evaluate derivative of bezier_path at given points

Parameters

t	points at which bezier_path is evaluated

14.3.3.2 deriv() [2/2]

```
template<typename T >
T libpath::bezier_path< T >::deriv (
            T t )  [inline]
```

evaluate derivative of bezier_path at given point

Parameters

t	point at which bezier_path is evaluated

14.3.3.3 eval() [1/2]

```
template<typename T >
std::vector<T> libpath::bezier_path< T >::eval (
            std::vector< T > t )
```

evaluate path approximated by bezier curve at given points

Parameters

t	points at which bezier_path is evaluated

14.3.3.4 eval() [2/2]

```
template<typename T >
T libpath::bezier_path< T >::eval (
            T t ) [inline]
```

evaluate path approximated by bezier curve at given point

Parameters

t	point at which bezier_path is evaluated

14.3.3.5 get_ctrl_pts()

```
template<typename T >
std::vector<T> libpath::bezier_path< T >::get_ctrl_pts ( ) const [inline]
```

get modified control points

Returns

modified control points

14.3.3.6 get_endtimes()

```
template<typename T >
std::tuple<T, T> libpath::bezier_path< T >::get_endtimes ( ) const  [inline]
```

get initial and final time of path

Returns

tuple of initial and final time of path

14.3.3.7 get_scaler()

```
template<typename T >
T libpath::bezier_path< T >::get_scaler ( ) const  [inline]
```

get the second scaler parameter

Returns

second scaler parameter

14.3.3.8 is_vaild()

```
template<typename T >
bool libpath::bezier_path< T >::is_vaild ( ) const  [inline]
```

check whether or not the bezier curve is valid to approximate path

Returns

vaildity of bezier curve

14.3.3.9 update()

```
template<typename T >
void libpath::bezier_path< T >::update (
            bezier< T > B_ )  [inline]
```

update bezier curve

Parameters

B↩ _↩	bezier curve to update

The documentation for this class was generated from the following file:

- libpath/bezier_path.hpp

14.4 libpath::fourier< T > Class Template Reference

Class which defines fourier function.

```
#include <fourier.hpp>
```

Public Member Functions

- fourier ()
  initialize fourier class
- fourier (unsigned int num_fourier, T period, std::vector< T > c)
  initialize fourier class
- fourier (const fourier< T > &copy)
  copy constructer of fourier class
- fourier< T > & operator= (const fourier< T > &copy)
  overloading of assignment operator for fourier class
- unsigned int terms (unsigned int n_f) const
  Inform how many terms need to define fourier function.
- std::vector< T > get_coeff () const
  get fourier coefficients
- void update (std::vector< T > c)
  update coefficients
- T eval (T t)
  evaluate the fourier function
- std::vector< T > eval (std::vector< T > t)
  evaluate the fourier function
- T deriv (T t)
  evaluate the derivative of fourier function
- std::vector< T > deriv (std::vector< T > t)
  evaluate the derivative of fourier function
- T nderiv (unsigned int n, T t)
  evaluate the nth derivative of fourier function
- std::vector< T > nderiv (unsigned int n, std::vector< T > t)
  evaluate the nth derivative of fourier function

14.4.1 Detailed Description

template<typename T>
class libpath::fourier< T >

Class which defines fourier function.

$$\phi\{c,T\}(t) = \sum_{i=0}^{n_f-1} c_{2i} \sin\left(\frac{2i\pi}{T}t\right) + c_{2i+1} \cos\left(\frac{2i\pi}{T}t\right) \tag{14.5}$$

, where n_f is the number of sine and cosine function used in fourier function, c is the weight and T is the period of fourier function.

Parameters

T	precision should be one of float, double, and long double

14.4.2 Constructor & Destructor Documentation

14.4.2.1 fourier()

```
template<typename T >
libpath::fourier< T >::fourier (
            unsigned int num_fourier,
            T period,
            std::vector< T > c )  [inline]
```

initialize fourier class

Parameters

num_fourier	number of sine and consine function to add
period	period of fourier function
c	coefficients of fourier function

14.4.3 Member Function Documentation

14.4.3.1 deriv() [1/2]

```
template<typename T >
std::vector<T> libpath::fourier< T >::deriv (
            std::vector< T > t )
```

evaluate the derivative of fourier function

Parameters

t	points to evaluate the derivative of fourier function

Returns

values of the derivative of fourier function evaluated at t

14.4.3.2 deriv() [2/2]

```
template<typename T >
T libpath::fourier< T >::deriv (
            T t )
```

evaluate the derivative of fourier function

Parameters

t	points to evaluate the derivative of fourier function

Returns

values of the derivative of fourier function evaluated at t

14.4.3.3 eval() [1/2]

```
template<typename T >
std::vector<T> libpath::fourier< T >::eval (
            std::vector< T > t )
```

evaluate the fourier function

Parameters

t	points to evaluate the fourier function

Returns

values of the fourier function evaluated at t

14.4.3.4 eval() [2/2]

```
template<typename T >
T libpath::fourier< T >::eval (
            T t )
```

evaluate the fourier function

Parameters

t	points to evaluate the fourier function

Returns

values of the fourier function evaluated at t

14.4.3.5 get_coeff()

```
template<typename T >
std::vector<T> libpath::fourier< T >::get_coeff ( ) const [inline]
```

get fourier coefficients

Returns

fourier coefficients

14.4.3.6 nderiv() [1/2]

```
template<typename T >
std::vector<T> libpath::fourier< T >::nderiv (
            unsigned int n,
            std::vector< T > t )
```

evaluate the nth derivative of fourier function

Parameters

n	order of derivative to compute
t	points to evaluate the derivative of fourier function

Returns

values of the nth order derivative of fourier function evaluated at t

14.4.3.7 nderiv() [2/2]

```
template<typename T >
T libpath::fourier< T >::nderiv (
            unsigned int n,
            T t )
```

evaluate the nth derivative of fourier function

Parameters

n	order of derivative to compute
t	points to evaluate the derivative of fourier function

Returns

values of the nth order derivative of fourier function evaluated at t

14.4.3.8 terms()

```
template<typename T >
unsigned int libpath::fourier< T >::terms (
            unsigned int n_f ) const [inline]
```

Inform how many terms need to define fourier function.

Parameters

n_f	number of fourier function

Returns

number of terms need to define fourier function

14.4.3.9 update()

```
template<typename T >
void libpath::fourier< T >::update (
            std::vector< T > c ) [inline]
```

update coefficients

Parameters

c	coefficients of fourier function

The documentation for this class was generated from the following file:

- libpath/fourier.hpp

14.5 libpath::fourier_path< T > Class Template Reference

Class for the path approximated by fourier function.

```
#include <fourier_path.hpp>
```

Public Member Functions

- fourier_path ()
  initalize fourier path class
- fourier_path (T t_init, T t_fin, T init, T fin)
  initalize fourier path class
- fourier_path (T t_init, T t_fin, T init, T fin, fourier< T > fourier)
  initalize fourier path class
- fourier_path (const fourier_path< T > &copy)
  copy constructor of fourier path class
- fourier_path< T > & operator= (const fourier_path< T > &copy)
  overloading of assignment operator for fourier_path class
- void update (fourier< T > fourier)
  update fourier function It also updates the validity of fourier function
- bool is_vaild () const
  check whether or not the fourier function is valid to approximate path
- std::tuple< T, T > get_endtimes () const
  get initial and final time of path
- T get_adder () const
  get adder
- T get_scaler () const
  get scaler
- std::vector< T > get_coeff () const
  get fourier coefficients
- T eval (T t)
  evaluate the path
- std::vector< T > eval (std::vector< T > t)
  evaluate the path
- T deriv (T t)
  evaluate the derivative of path
- std::vector< T > deriv (std::vector< T > t)
  evaluate the path
- T nderiv (unsigned int n, T t)
  evaluate the nth derivative of path
- std::vector< T > nderiv (unsigned int n, std::vector< T > t)
  evaluate the nth derivative of path

14.5.1 Detailed Description

template<typename T>
class libpath::fourier_path< T >

Class for the path approximated by fourier function.

$$\Psi\{\phi\{c,T\}(t)\}(t) = a + s\phi\{c,T\}(t) \tag{14.6}$$

,where $\phi\{c,T\}(t)$ is a fourier function defined in libpath::fourier , a and s are adder and scaler to match boundary condition $\Psi(t_0) = p_0, \Psi(t_1) = p_1$, respectively.

Parameters

T	precision should be one of float, double, long double

Warning

If your fourier function is not vaild for the approximation of path, eval and deriv method return always zero

14.5.2 Constructor & Destructor Documentation

14.5.2.1 fourier_path() [1/2]

```
template<typename T >
libpath::fourier_path< T >::fourier_path (
            T t_init,
            T t_fin,
            T init,
            T fin )  [inline]
```

initalize fourier path class

Parameters

t_init	initial time
t_fin	finial time
init	initial value of path
fin	finial value of path

14.5.2.2 fourier_path() [2/2]

```
template<typename T >
libpath::fourier_path< T >::fourier_path (
```

```
            T t_init,
            T t_fin,
            T init,
            T fin,
            fourier< T > fourier )  [inline]
```

initalize fourier path class

Parameters

t_init	initial time
t_fin	finial time
init	initial value of path
fin	finial value of path
fourier	fourier function used to approximation of path

14.5.3 Member Function Documentation

14.5.3.1 deriv() [1/2]

```
template<typename T >
std::vector<T> libpath::fourier_path< T >::deriv (
            std::vector< T > t )
```

evaluate the path

Parameters

t	points to evaluate the path

Returns

the path evaluated at t

14.5.3.2 deriv() [2/2]

```
template<typename T >
T libpath::fourier_path< T >::deriv (
            T t )  [inline]
```

evaluate the derivative of path

Parameters

t	points to evaluate

Returns

the derivative of path evaluated at t

14.5.3.3 eval() [1/2]

```
template<typename T >
std::vector<T> libpath::fourier_path< T >::eval (
            std::vector< T > t )
```

evaluate the path

Parameters

t	points to evaluate the path

Returns

the path evaluated at t

14.5.3.4 eval() [2/2]

```
template<typename T >
T libpath::fourier_path< T >::eval (
            T t )  [inline]
```

evaluate the path

Parameters

t	points to evaluate the path

Returns

the path evaluated at t

14.5.3.5 get_adder()

```
template<typename T >
T libpath::fourier_path< T >::get_adder ( ) const [inline]
```

get adder

Returns

adder

14.5.3.6 get_coeff()

```
template<typename T >
std::vector<T> libpath::fourier_path< T >::get_coeff ( ) const [inline]
```

get fourier coefficients

Returns

fourier coefficients

14.5.3.7 get_endtimes()

```
template<typename T >
std::tuple<T, T> libpath::fourier_path< T >::get_endtimes ( ) const [inline]
```

get initial and final time of path

Returns

tuple of initial and final time of path

14.5.3.8 get_scaler()

```
template<typename T >
T libpath::fourier_path< T >::get_scaler ( ) const [inline]
```

get scaler

Returns

scaler

14.5.3.9 is_vaild()

```
template<typename T >
bool libpath::fourier_path< T >::is_vaild ( ) const  [inline]
```

check whether or not the fourier function is valid to approximate path

Returns

vaildity of fourier function

14.5.3.10 nderiv() [1/2]

```
template<typename T >
std::vector<T> libpath::fourier_path< T >::nderiv (
            unsigned int n,
            std::vector< T > t )
```

evaluate the nth derivative of path

Parameters

n	order of derivative to compute
t	points to evaluate

Returns

the nth order derivative of path evaluated at t

14.5.3.11 nderiv() [2/2]

```
template<typename T >
T libpath::fourier_path< T >::nderiv (
            unsigned int n,
            T t )  [inline]
```

evaluate the nth derivative of path

Parameters

n	order of derivative to compute
t	points to evaluate

Returns

the nth order derivative of path evaluated at t

14.5.3.12 update()

```
template<typename T >
void libpath::fourier_path< T >::update (
            fourier< T > fourier )  [inline]
```

update fourier function It also updates the validity of fourier function

Parameters

fourier	fourier function to update

The documentation for this class was generated from the following file:

- libpath/fourier_path.hpp

14.6 libpath::gau_kron_table< T, N > Class Template Reference

table for gauss kronrod node and weights

```
#include <node_weight_table.hpp>
```

Public Attributes

- unsigned int order

    order of gauss-kronrod quadrature;
- std::vector< T > nodes

    node of gauss-kronrod quadrature;
- std::vector< T > weight_gauss

    weight of gauss quadrature;
- std::vector< T > weight_kronrod

    weight of kronrod quadrature;

14.6.1 Detailed Description

template<typename T, unsigned int N>
class libpath::gau_kron_table< T, N >

table for gauss kronrod node and weights

Parameters

T	precision should be one of float, double, long double
N	order of gauss-kronrod quadrature currently only supports N=15, 21, 31, 41, 51, 61

Note

coefficients are obtained from `gau-kronrod-nodes-weights`

The documentation for this class was generated from the following file:

- libpath/node_weight_table.hpp

14.7 kepler_lag< T > Class Template Reference

functor class for the kepler lagranian

Public Member Functions

- T operator() (T t, vector< T > p, vector< T > dp) const
  evaluates kepler lagrangian at given time

14.7.1 Detailed Description

template<typename T>
class kepler_lag< T >

functor class for the kepler lagranian

Parameters

T	precision should be the one of float, double, long double.

14.7.2 Member Function Documentation

14.7.2.1 operator()()

```
template<typename T >
T kepler_lag< T >::operator() (
                T t,
```

```
vector< T > p,
vector< T > dp ) const  [inline]
```

evaluates kepler lagrangian at given time

Parameters

t	time
p	path
dp	derivative of path

Returns

lagrangian evaluated at given time

The documentation for this class was generated from the following file:

- main.cpp

14.8 libmcm::mcm< T, Basis, Path, Lag > Class Template Reference

class to Minimize the action via Monte Carlo Metropolis Method. It uses mt19937 random number generator to generates distribution. Moreover it samples path via random walk. To make random walk, it samples real number from normal distribution and move path by the sampled real number.

```
#include <mcm.hpp>
```

Public Member Functions

- mcm ()
  initialize mcm class
- mcm (T t_0, T t_1, std::vector< T > p_0, std::vector< T > p_1, T rel_tol, unsigned int order_)
  initialize mcm class
- mcm (T t_0, T t_1, std::vector< T > p_0, std::vector< T > p_1, T rel_tol, unsigned int order_, T add_setup_)
  initialize mcm class
- mcm (const mcm< T, Basis, Path, Lag > &copy)
  copy constructer of mcm class
- mcm< T, Basis, Path, Lag > & operator= (const mcm< T, Basis, Path, Lag > &copy)
  overloading of assignment operator for mcm class
- void set_init_guess (std::vector< std::vector< T >> init_c)
  set initial guess
- void set_init_guess ()
  set initial guess randomly
- T get_init_action (T &e)
  get action of initial guess
- std::vector< Path > get_init_path () const
  get initial path
- std::vector< T > init_eval (T t)
  evaluate initial path at given t
- std::vector< std::vector< T > > init_eval (std::vector< T > t)
  evaluate initial path at given t
- T get_min_action (T &e)
  get action of minimum guess

- std::vector< Path > get_min_path () const

    get minimum path
- std::vector< T > min_eval (T t)

    evaluate minimum path at given t
- std::vector< std::vector< T > > min_eval (std::vector< T > t)

    evaluate minium path at given t
- std::tuple< std::size_t, T > optimize (std::size_t num_iter, T step_size, T lambda)

    minimize the action via Monte Carlo Metropolis method
- std::tuple< std::size_t, T > optimize (std::size_t num_iter, T step_size, T lambda, std::string monitor)

    minimize the action via Monte Carlo Metropolis method

14.8.1 Detailed Description

template<typename T, typename Basis, typename Path, typename Lag>
class libmcm::mcm< T, Basis, Path, Lag >

class to Minimize the action via Monte Carlo Metropolis Method. It uses mt19937 random number generator to generates distribution. Moreover it samples path via random walk. To make random walk, it samples real number from normal distribution and move path by the sampled real number.

Parameters

T	precision should be one of float, double and long double
Basis	type of basis used to approximate path
Path	type of path
Lag	lagrangian of action functor class which has time, path and derivative of path as variable and it returns value of lagranian at given time

See also

Monte Carlo Metropolis method

Note

This is abstract class use libmcm::mcm_fourier (for libpath::fourier_path) or libmcm::mcm_bezier (for libpath::bezier_path), instead.

14.8.2 Constructor & Destructor Documentation

14.8.2.1 mcm() [1/2]

```
template<typename T , typename Basis , typename Path , typename Lag >
libmcm::mcm< T, Basis, Path, Lag >::mcm (
            T t_0,
            T t_1,
            std::vector< T > p_0,
            std::vector< T > p_1,
            T rel_tol,
            unsigned int order_ )  [inline]
```

initialize mcm class

Parameters

t_0	initial time
t_1	finial time
p_0	value of path at initial time
p_1	value of path at finial time
rel_tol	relative tolerance for action integral
*order_*	order of basis function

14.8.2.2 mcm() [2/2]

```
template<typename T , typename Basis , typename Path , typename Lag >
libmcm::mcm< T, Basis, Path, Lag >::mcm (
            T t_0,
            T t_1,
            std::vector< T > p_0,
            std::vector< T > p_1,
            T rel_tol,
            unsigned int order_,
            T add_setup_ )  [inline]
```

initialize mcm class

Parameters

t_0	initial time
t_1	finial time
p_0	value of path at initial time
p_1	value of path at finial time
rel_tol	relative tolerance for action integral
*order_*	order of basis function
*add_setup_*	additional setup used to define basis function

14.8.3 Member Function Documentation

14.8.3.1 get_init_action()

```
template<typename T , typename Basis , typename Path , typename Lag >
T libmcm::mcm< T, Basis, Path, Lag >::get_init_action (
            T & e )
```

get action of initial guess

Parameters

out	*e*	estimated error of action integral

Returns

action of initial guess

See also

libpath::action

14.8.3.2 get_init_path()

```
template<typename T , typename Basis , typename Path , typename Lag >
std::vector<Path> libmcm::mcm< T, Basis, Path, Lag >::get_init_path ( ) const  [inline]
```

get initial path

Returns

initial_path

14.8.3.3 get_min_action()

```
template<typename T , typename Basis , typename Path , typename Lag >
T libmcm::mcm< T, Basis, Path, Lag >::get_min_action (
            T & e )
```

get action of minimum guess

Parameters

out	*e*	estimated error of action integral

Returns

action of minimum guess

See also

libpath::action

14.8.3.4 get_min_path()

```
template<typename T , typename Basis , typename Path , typename Lag >
std::vector<Path> libmcm::mcm< T, Basis, Path, Lag >::get_min_path ( ) const  [inline]
```

get minimum path

Returns

minimum path

14.8.3.5 init_eval() [1/2]

```
template<typename T , typename Basis , typename Path , typename Lag >
std::vector<std::vector<T> > libmcm::mcm< T, Basis, Path, Lag >::init_eval (
            std::vector< T > t )
```

evaluate initial path at given t

Parameters

t	time to evaluate initial path

Returns

initial path evaluated at t

14.8.3.6 init_eval() [2/2]

```
template<typename T , typename Basis , typename Path , typename Lag >
std::vector<T> libmcm::mcm< T, Basis, Path, Lag >::init_eval (
            T t )
```

evaluate initial path at given t

Parameters

t	time to evaluate initial path

Returns

initial path evaluated at t

14.8.3.7 min_eval() [1/2]

```
template<typename T , typename Basis , typename Path , typename Lag >
std::vector<std::vector<T> > libmcm::mcm< T, Basis, Path, Lag >::min_eval (
            std::vector< T > t )
```

evaluate minium path at given t

Parameters

t	time to evaluate minimum path

Returns

minimum path evaluated at t

14.8.3.8 min_eval() [2/2]

```
template<typename T , typename Basis , typename Path , typename Lag >
std::vector<T> libmcm::mcm< T, Basis, Path, Lag >::min_eval (
            T t )
```

evaluate minimum path at given t

Parameters

t	time to evaluate minimum path

Returns

minimum path evaluated at t

14.8.3.9 optimize() [1/2]

```
template<typename T , typename Basis , typename Path , typename Lag >
std::tuple<std::size_t, T> libmcm::mcm< T, Basis, Path, Lag >::optimize (
            std::size_t num_iter,
            T step_size,
            T lambda )
```

minimize the action via Monte Carlo Metropolis method

Parameters

num_iter	number of iteration
step_size	step size of random walk
lambda	parameter which controls acceptance of move

Returns

tuple of number of accepted move and acceptance ratio.

14.8.3.10 optimize() [2/2]

```
template<typename T , typename Basis , typename Path , typename Lag >
std::tuple<std::size_t, T> libmcm::mcm< T, Basis, Path, Lag >::optimize (
            std::size_t num_iter,
            T step_size,
            T lambda,
            std::string monitor )
```

minimize the action via Monte Carlo Metropolis method

Parameters

num_iter	number of iteration
step_size	step size of random walk
lambda	parameter which controls acceptance of move
monitor	filename to monitor optimization process saved file has c++ binary format. It stores twice of number of accepted move T type data as $a, e, a, e, a, e, \ldots$, where a is the action of i th accepted move and e is the estimated error of i th action integration.

Returns

tuple of number of accepted move and acceptance ratio.

14.8.3.11 set_init_guess()

```
template<typename T , typename Basis , typename Path , typename Lag >
void libmcm::mcm< T, Basis, Path, Lag >::set_init_guess (
            std::vector< std::vector< T >> init_c )
```

set initial guess

Parameters

init←_c	initial coefficients

The documentation for this class was generated from the following file:

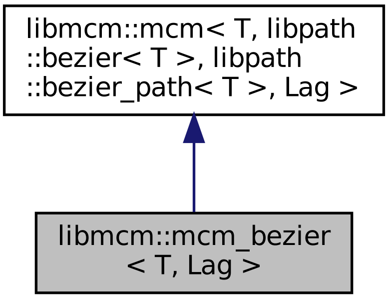
- libmcm/mcm.hpp

14.9 libmcm::mcm_bezier< T, Lag > Class Template Reference

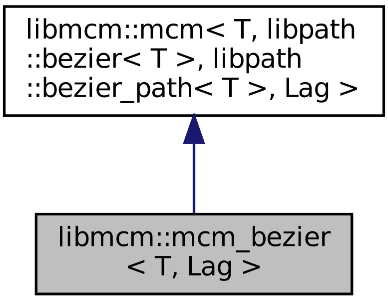
Derivated class of libmcm::mcm class for path approximated by bezier curve.

```
#include <mcm_bezier.hpp>
```

Inheritance diagram for libmcm::mcm_bezier< T, Lag >:



Collaboration diagram for libmcm::mcm_bezier< T, Lag >:



Public Member Functions

- mcm_bezier ()

  initalize mcm_bezier class
- mcm_bezier (T t_0, T t_1, std::vector< T > p_0, std::vector< T > p_1, T rel_tol, unsigned int n_)

  initalize mcm_bezier class
- mcm_bezier (const mcm_bezier< T, Lag > &copy)

  copy constructor of mcm_bezier class
- mcm_bezier< T, Lag > & operator= (const mcm_bezier< T, Lag > &copy)

overloading of assignment operator for mcm_bezier class

- std::tuple< std::vector< T >, std::vector< std::vector< T > > > get_init_coeff ()

  get scaler and coefficients of initial guess

- std::tuple< std::vector< T >, std::vector< std::vector< T > > > get_min_coeff ()

  get scaler and coefficients of minimal guess

14.9.1 Detailed Description

template<typename T, typename Lag>
class libmcm::mcm_bezier< T, Lag >

Derivated class of libmcm::mcm class for path approximated by bezier curve.

Parameters

T	precision should be one of float, double, or long double
Lag	lagrangian functor class which has time, path and derivative of path as variable and it returns value of lagranian at given time

See also

libpath::bezier_path class

libmcm::mcm class

14.9.2 Constructor & Destructor Documentation

14.9.2.1 mcm_bezier()

```
template<typename T , typename Lag >
libmcm::mcm_bezier< T, Lag >::mcm_bezier (
            T t_0,
            T t_1,
            std::vector< T > p_0,
            std::vector< T > p_1,
            T rel_tol,
            unsigned int n_ )  [inline]
```

initalize mcm_bezier class

Parameters

t_0	initial time
t_1	final time
p_0	value of path at initial time
p_1	vale of path at final time
rel_tol	relative tolerance for action integral
*n_*	order of bezier curve

14.9.3 Member Function Documentation

14.9.3.1 get_init_coeff()

```
template<typename T , typename Lag >
std::tuple<std::vector<T>, std::vector<std::vector<T> > > libmcm::mcm_bezier< T, Lag >↩
::get_init_coeff ( )
```

get scaler and coefficients of initial guess

Returns

tuple of scaler and coefficients of initial guess

14.9.3.2 get_min_coeff()

```
template<typename T , typename Lag >
std::tuple<std::vector<T>, std::vector<std::vector<T> > > libmcm::mcm_bezier< T, Lag >↩
::get_min_coeff ( )
```

get scaler and coefficients of minimal guess

Returns

tuple of scaler and coefficients of minimal guess

The documentation for this class was generated from the following file:

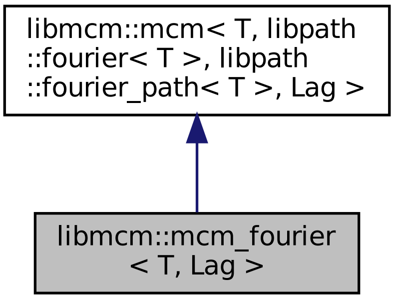
- libmcm/mcm_bezier.hpp

14.10 libmcm::mcm_fourier< T, Lag > Class Template Reference

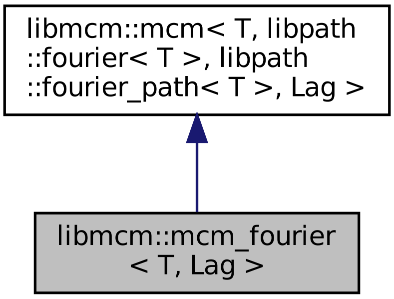
Derivated class of libmcm::mcm class for path approximated by fourier function.

```
#include <mcm_fourier.hpp>
```

Inheritance diagram for libmcm::mcm_fourier< T, Lag >:



Collaboration diagram for libmcm::mcm_fourier< T, Lag >:



Public Member Functions

- mcm_fourier ()

  initalize mcm_fourier class
- mcm_fourier (T t_0, T t_1, std::vector< T > p_0, std::vector< T > p_1, T rel_tol, unsigned int n_f, T period)

  initalize mcm_fourier class
- mcm_fourier (const mcm_fourier< T, Lag > &copy)

  copy constructor of mcm_fourier class
- mcm_fourier< T, Lag > & operator= (const mcm_fourier< T, Lag > &copy)

  overloading of assignment operator for mcm_fourier class
- std::tuple< std::vector< T >, std::vector< T >, std::vector< std::vector< T > > > get_init_coeff ()

  get adder, scaler and coefficients of initial guess
- std::tuple< std::vector< T >, std::vector< T >, std::vector< std::vector< T > > > get_min_coeff ()

  get adder, scaler and coefficients of minimal guess

14.10.1 Detailed Description

template<typename T, typename Lag>
class libmcm::mcm_fourier< T, Lag >

Derivated class of libmcm::mcm class for path approximated by fourier function.

Parameters

T	precision should be one of float, double, or long double
Lag	lagrangian functor class which has time, path and derivative of path as variable and it returns value of lagranian at given time

See also

libpath::fourier_path class

libmcm::mcm class

14.10.2 Constructor & Destructor Documentation

14.10.2.1 mcm_fourier()

```
template<typename T , typename Lag >
libmcm::mcm_fourier< T, Lag >::mcm_fourier (
            T t_0,
            T t_1,
            std::vector< T > p_0,
            std::vector< T > p_1,
            T rel_tol,
            unsigned int n_f,
            T period )  [inline]
```

initalize mcm_fourier class

Parameters

t_0	initial time
t_1	final time
p_0	value of path at initial time
p_1	vale of path at final time
rel_tol	relative tolerance for action integral
n_f	number of fourier function to use
period	period of fourier function

14.10.3 Member Function Documentation

14.10.3.1 get_init_coeff()

```
template<typename T , typename Lag >
std::tuple<std::vector<T>, std::vector<T>, std::vector<std::vector<T> > > libmcm::mcm_fourier<
T, Lag >::get_init_coeff ( )
```

get adder, scaler and coefficients of initial guess

Returns

tuple of adder, scaler and coefficients of initial guess

14.10.3.2 get_min_coeff()

```
template<typename T , typename Lag >
std::tuple<std::vector<T>, std::vector<T>, std::vector<std::vector<T> > > libmcm::mcm_fourier<
T, Lag >::get_min_coeff ( )
```

get adder, scaler and coefficients of minimal guess

Returns

tuple of adder, scaler and coefficients of minimal guess

The documentation for this class was generated from the following file:

- libmcm/mcm_fourier.hpp

Chapter 15

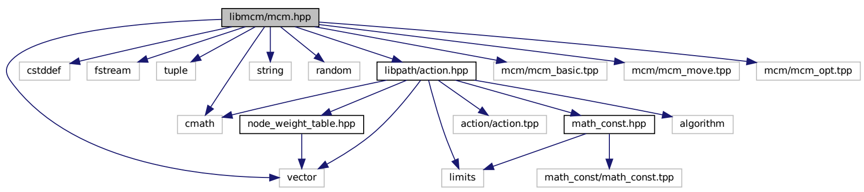
File Documentation

15.1 libmcm/mcm.hpp File Reference

headerfile for the minimization of the action by Monte Carlo Metropolis method described in `Entropy 2020, 22(9), 916`

```
#include <cmath>
#include <cstddef>
#include <fstream>
#include <tuple>
#include <vector>
#include <string>
#include <random>
#include "libpath/action.hpp"
#include "mcm/mcm_basic.tpp"
#include "mcm/mcm_move.tpp"
#include "mcm/mcm_opt.tpp"
```

Include dependency graph for mcm.hpp:



Classes

- class libmcm::mcm< T, Basis, Path, Lag >

    class to Minimize the action via Monte Carlo Metropolis Method. It uses mt19937 random number generator to generates distribution. Moreover it samples path via random walk. To make random walk, it samples real number from normal distribution and move path by the sampled real number.

15.1.1 Detailed Description

headerfile for the minimization of the action by Monte Carlo Metropolis method described in `Entropy 2020, 22(9), 916`

Author

pistack (Junho Lee)

Date

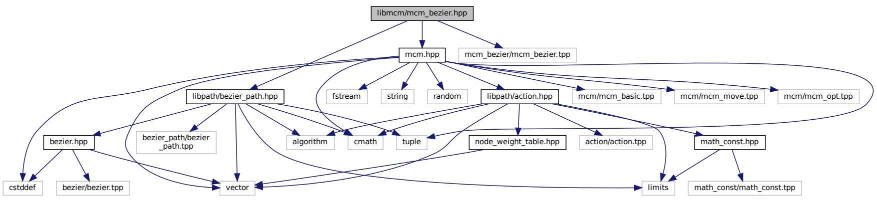
2021. 11. 13.

15.2 libmcm/mcm_bezier.hpp File Reference

headerfile for mcm_bezier class which is derivated by libmcm::mcm class

```
#include "libpath/bezier_path.hpp"
#include "mcm.hpp"
#include "mcm_bezier/mcm_bezier.tpp"
```

Include dependency graph for mcm_bezier.hpp:



Classes

- class libmcm::mcm_bezier< T, Lag >

  Derivated class of libmcm::mcm class for path approximated by bezier curve.

15.2.1 Detailed Description

headerfile for mcm_bezier class which is derivated by libmcm::mcm class

Author

pistack (Junho Lee)

Date

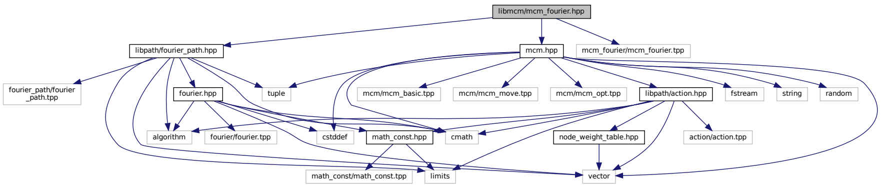
2021. 11. 13.

15.3 libmcm/mcm_fourier.hpp File Reference

headerfile for mcm_fourier class which is derivated by libmcm::mcm class

```
#include "libpath/fourier_path.hpp"
#include "mcm.hpp"
#include "mcm_fourier/mcm_fourier.tpp"
```

Include dependency graph for mcm_fourier.hpp:



Classes

- class libmcm::mcm_fourier< T, Lag >

  Derivated class of libmcm::mcm class for path approximated by fourier function.

15.3.1 Detailed Description

headerfile for mcm_fourier class which is derivated by libmcm::mcm class

Author

pistack (Junho Lee)

Date

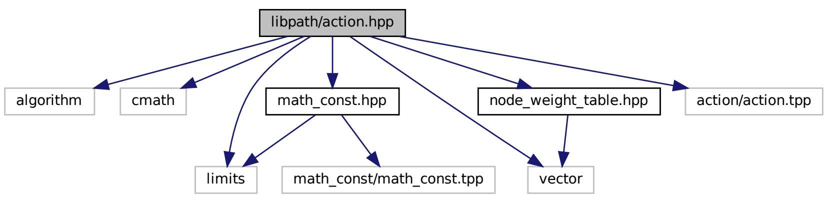
2021. 11. 13.

15.4 libpath/action.hpp File Reference

header file for evaluation of the action

```
#include <algorithm>
#include <cmath>
#include <limits>
#include <vector>
#include "math_const.hpp"
#include "node_weight_table.hpp"
#include "action/action.tpp"
```

Include dependency graph for action.hpp:



Classes

- class libpath::action< T, Path, Lag >

  class which computes action

15.4.1 Detailed Description

header file for evaluation of the action

Author

pistack (Junho Lee)

Date

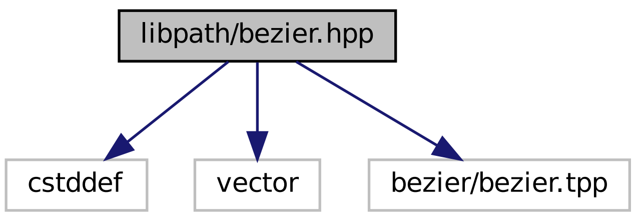
2021. 11. 15.

15.5 libpath/bezier.hpp File Reference

headerfile for bezier curve and path approximated by bezier curve

```
#include <cstddef>
#include <vector>
#include "bezier/bezier.tpp"
```

Include dependency graph for bezier.hpp:



Classes

- class libpath::bezier< T >

  Class which defines n th order Bezier curve.

15.5.1 Detailed Description

headerfile for bezier curve and path approximated by bezier curve

Author

pistack (Junho Lee)

Date

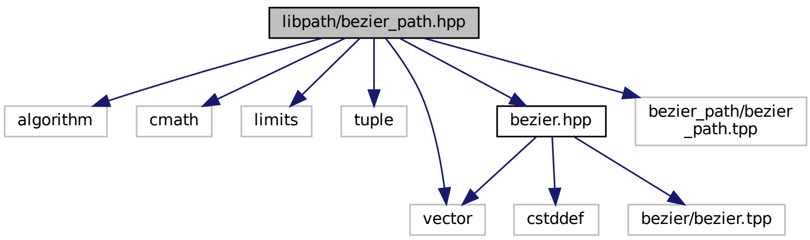
2021. 11. 12.

15.6 libpath/bezier_path.hpp File Reference

headerfile for path approximated by bezier curve

```
#include <algorithm>
#include <cmath>
#include <limits>
#include <tuple>
#include <vector>
#include "bezier.hpp"
#include "bezier_path/bezier_path.tpp"
```

Include dependency graph for bezier_path.hpp:



Classes

- class libpath::bezier_path< T >

  Class which approximate path by n th order Bezier curve.

15.6.1 Detailed Description

headerfile for path approximated by bezier curve

Author

pistack (Junho Lee)

Date

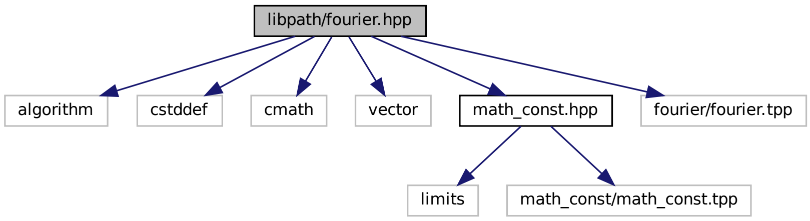
2021. 11. 15.

15.7 libpath/fourier.hpp File Reference

headerfile for fourier function

```
#include <algorithm>
#include <cstddef>
#include <cmath>
#include <vector>
#include "math_const.hpp"
#include "fourier/fourier.tpp"
```

Include dependency graph for fourier.hpp:



Classes

- class libpath::fourier< T >

    Class which defines fourier function.

15.7.1 Detailed Description

headerfile for fourier function

Author

pistack (Junho Lee)

Date

2021. 11. 15.

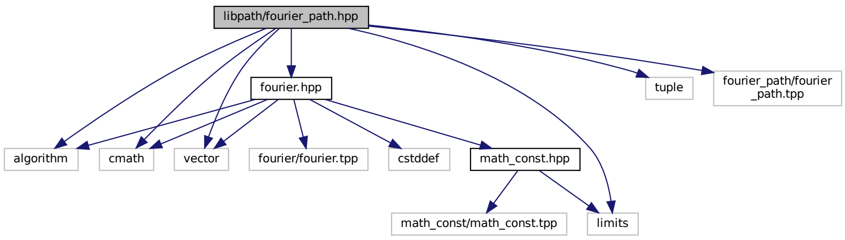
15.8 libpath/fourier_path.hpp File Reference

headerfile for path approximated by fourier function

```
#include <algorithm>
#include <cmath>
#include <tuple>
#include <vector>
#include <limits>
```

```
#include "fourier.hpp"
#include "fourier_path/fourier_path.tpp"
```

Include dependency graph for fourier_path.hpp:



Classes

- class libpath::fourier_path< T >

    Class for the path approximated by fourier function.

15.8.1 Detailed Description

headerfile for path approximated by fourier function

Author

pistack (Junho Lee)

Date

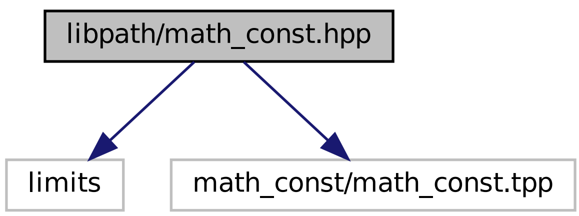
2021. 11. 10.

15.9 libpath/math_const.hpp File Reference

provides mathematical constent with different precision

```
#include <limits>
#include "math_const/math_const.tpp"
```

Include dependency graph for math_const.hpp:



Functions

- template<typename T >
  constexpr T libpath::PI ()

  provides pi
- template<typename T >
  constexpr T libpath::EXP1 ()

  provides exponential constant

15.9.1 Detailed Description

provides mathematical constent with different precision

Author

pistack (Junho Lee)

Date

2021. 11. 15.

15.9.2 Function Documentation

15.9.2.1 EXP1()

```
template<typename T >
constexpr T libpath::EXP1 ( )  [constexpr]
```

provides exponential constant

Parameters

T	precision should be one of float, double or long double

15.9.2.2 PI()

```
template<typename T >
constexpr T libpath::PI ( )  [constexpr]
```

provides pi

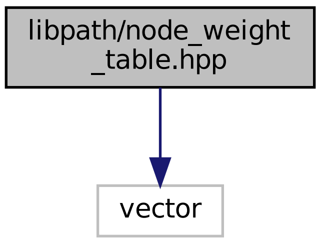
Parameters

T	precision should be one of float, double or long double

15.10 libpath/node_weight_table.hpp File Reference

table for node and weights

`#include <vector>`
Include dependency graph for node_weight_table.hpp:



Classes

- class libpath::gau_kron_table< T, N >
  table for gauss kronrod node and weights

15.10.1 Detailed Description

table for node and weights

Author

pistack (Junho Lee)

Date

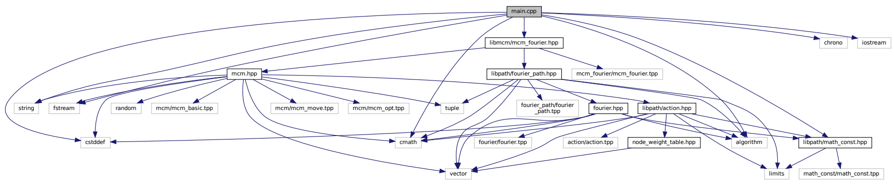
2021. 11. 12.

15.11 main.cpp File Reference

main program for homework3 of Computer1 class in Yonsei University Interactively reads inital condition, order of basis function, number of points to evaluate, number of iteration, step size, parameter lambda and output file name then computes and saves solution.

```
#include <algorithm>
#include <cmath>
#include <cstddef>
#include <chrono>
#include <string>
#include <iostream>
#include <fstream>
#include "libpath/math_const.hpp"
#include "libmcm/mcm_fourier.hpp"
```

Include dependency graph for main.cpp:



Classes

- class kepler_lag< T >

    functor class for the kepler lagranian

Macros

- #define **PATH_TYPE_FOURIER**
- #define **PRECISION_LEVEL** 1
- #define **MONITOR** 0
- #define **PRECISION** double
- #define **DIGITS** 14

Functions

- int **main** (void)

Variables

- constexpr PRECISION **pi** = libpath::PI<PRECISION>()

15.11.1 Detailed Description

main program for homework3 of Computer1 class in Yonsei University Interactively reads inital condition, order of basis function, number of points to evaluate, number of iteration, step size, parameter lambda and output file name then computes and saves solution.

Author

pistack (Junho Lee)

Date

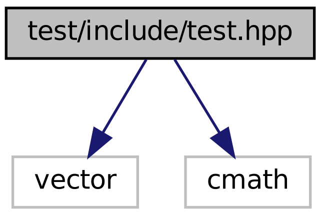
2021. 11. 15.

15.12 test/include/test.hpp File Reference

header file for testing libpath

```
#include <vector>
#include <cmath>
```

Include dependency graph for test.hpp:



Macros

- #define **PRECISION** float
- #define **DIGITS** 6

Functions

- int test::test_action_kepler ()

  test libpath::action::eval() routine with kepler lagrangian
- int test::test_action_simple ()

  test libpath::action::eval() routine with various simple lagrangian
- int test::test_action_vaildity ()

  test libpath::action::is_vaild() routine
- int test::test_bezier_path ()

  test libpath::bezier_path class
- int test::test_bezier ()

  test libpath::bezier class
- int test::test_fourier_path ()

  test libpath::fourier_path class
- int test::test_fourier ()

  test libpath::fourier class

15.12.1 Detailed Description

header file for testing libpath

Author

pistack (Junho Lee)

Date

2021. 11. 12.

15.12.2 Function Documentation

15.12.2.1 test_action_kepler()

`int test::test_action_kepler ( )`

test libpath::action::eval() routine with kepler lagrangian

Returns

test result

15.12.2.2 test_action_simple()

`int test::test_action_simple ( )`

test libpath::action::eval() routine with various simple lagrangian

Returns

test result

15.12.2.3 test_action_vaildity()

`int test::test_action_vaildity ( )`

test libpath::action::is_vaild() routine

Returns

test result

15.12.2.4 test_bezier()

```
int test::test_bezier ( )
```

test libpath::bezier class

Returns

test result

15.12.2.5 test_bezier_path()

```
int test::test_bezier_path ( )
```

test libpath::bezier_path class

Returns

test result

15.12.2.6 test_fourier()

```
int test::test_fourier ( )
```

test libpath::fourier class

Returns

test result

15.12.2.7 test_fourier_path()

```
int test::test_fourier_path ( )
```

test libpath::fourier_path class

Returns

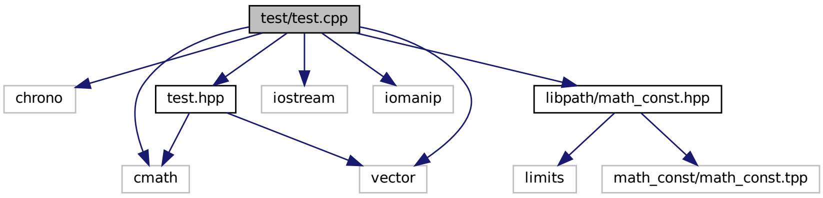
test result

15.13 test/test.cpp File Reference

test libpath

```
#include <chrono>
#include <cmath>
#include <vector>
#include <iostream>
#include <iomanip>
#include "libpath/math_const.hpp"
#include "test.hpp"
```

Include dependency graph for test.cpp:



Functions

- int **main** (void)

15.13.1 Detailed Description

test libpath

Author

pistack (Junho Lee)

Date

2021. 11. 15.

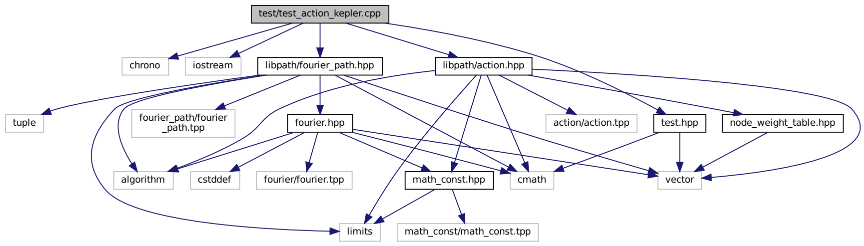
15.14 test/test_action_kepler.cpp File Reference

test action::eval() routine with kepler action

```
#include <chrono>
#include <iostream>
#include "libpath/fourier_path.hpp"
#include "libpath/action.hpp"
```

`#include "test.hpp"`

Include dependency graph for test_action_kepler.cpp:



Functions

- int test::test_action_kepler ()

    test libpath::action::eval() routine with kepler lagrangian

15.14.1 Detailed Description

test action::eval() routine with kepler action

Author

pistack (Junho Lee)

Date

2021. 11. 13.

15.14.2 Function Documentation

15.14.2.1 test_action_kepler()

```
int test::test_action_kepler ( )
```

test libpath::action::eval() routine with kepler lagrangian

Returns

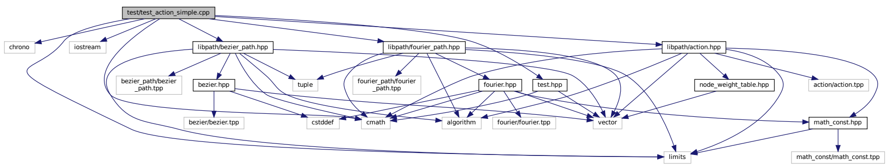
test result

15.15 test/test_action_simple.cpp File Reference

test action::eval() routine with simple function

```
#include <chrono>
#include <cmath>
#include <iostream>
#include <limits>
#include "libpath/bezier_path.hpp"
#include "libpath/fourier_path.hpp"
#include "libpath/action.hpp"
#include "test.hpp"
```

Include dependency graph for test_action_simple.cpp:



Functions

- int test::test_action_simple ()

    test libpath::action::eval() routine with various simple lagrangian

15.15.1 Detailed Description

test action::eval() routine with simple function

Author

pistack (Junho Lee)

Date

2021. 11. 13.

15.15.2 Function Documentation

15.15.2.1 test_action_simple()

```
int test::test_action_simple ( )
```

test libpath::action::eval() routine with various simple lagrangian

Returns

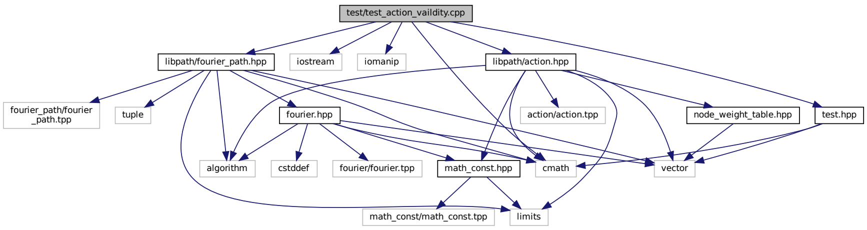
test result

15.16 test/test_action_vaildity.cpp File Reference

test action::is_vaild() routine

```
#include <cmath>
#include <iostream>
#include <iomanip>
#include "libpath/fourier_path.hpp"
#include "libpath/action.hpp"
#include "test.hpp"
```

Include dependency graph for test_action_vaildity.cpp:



Functions

- int test::test_action_vaildity ()
    test libpath::action::is_vaild() routine

15.16.1 Detailed Description

test action::is_vaild() routine

Author

pistack (Junho Lee)

Date

2021. 11. 12.

15.16.2 Function Documentation

15.16.2.1 test_action_vaildity()

`int test::test_action_vaildity ( )`

test libpath::action::is_vaild() routine

Returns

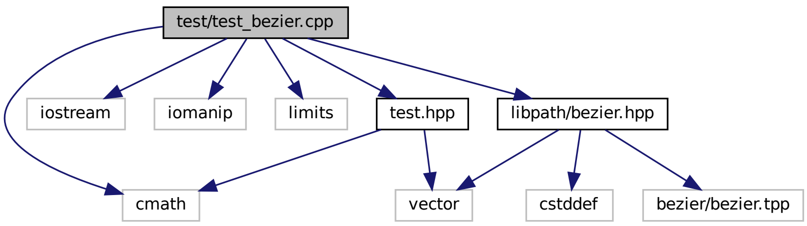
test result

15.17 test/test_bezier.cpp File Reference

test bezier class routine

```
#include <cmath>
#include <iostream>
#include <iomanip>
#include <limits>
#include "libpath/bezier.hpp"
#include "test.hpp"
```

Include dependency graph for test_bezier.cpp:



Functions

- int test::test_bezier ()

    test libpath::bezier class

15.17.1 Detailed Description

test bezier class routine

Author

pistack (Junho Lee)

Date

2021. 11. 12.

15.17.2 Function Documentation

15.17.2.1 test_bezier()

```
int test::test_bezier ( )
```

test libpath::bezier class

Returns

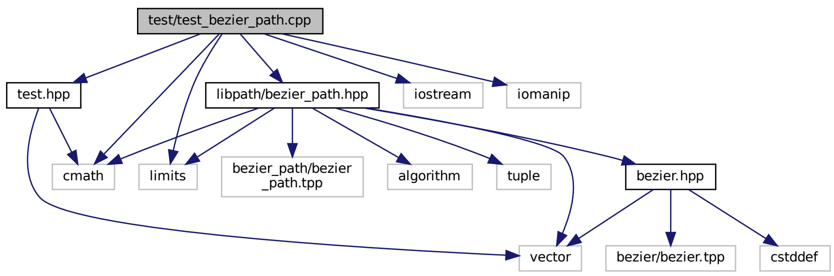
test result

15.18 test/test_bezier_path.cpp File Reference

test bezier_path class routine

```
#include <cmath>
#include <iostream>
#include <iomanip>
#include <limits>
#include "libpath/bezier_path.hpp"
#include "test.hpp"
```

Include dependency graph for test_bezier_path.cpp:



Functions

- int test::test_bezier_path ()

  test libpath::bezier_path class

15.18.1 Detailed Description

test bezier_path class routine

Author

pistack (Junho Lee)

Date

2021. 11. 12.

15.18.2 Function Documentation

15.18.2.1 test_bezier_path()

```
int test::test_bezier_path ( )
```

test libpath::bezier_path class

Returns

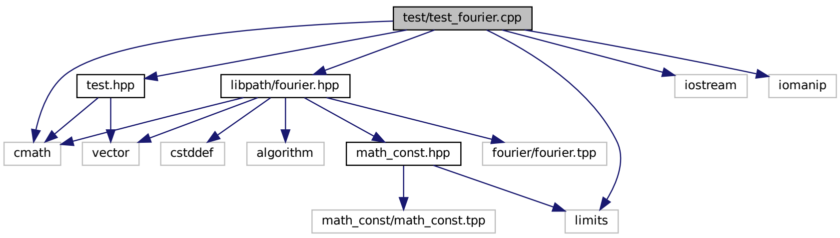
test result

15.19 test/test_fourier.cpp File Reference

test fourier class routine

```
#include <cmath>
#include <iostream>
#include <iomanip>
#include <limits>
#include "libpath/fourier.hpp"
#include "test.hpp"
```

Include dependency graph for test_fourier.cpp:



Functions

- int test::test_fourier ()

    test libpath::fourier class

15.19.1 Detailed Description

test fourier class routine

Author

pistack (Junho Lee)

Date

2021. 11. 12.

15.19.2 Function Documentation

15.19.2.1 test_fourier()

```
int test::test_fourier ( )
```

test libpath::fourier class

Returns

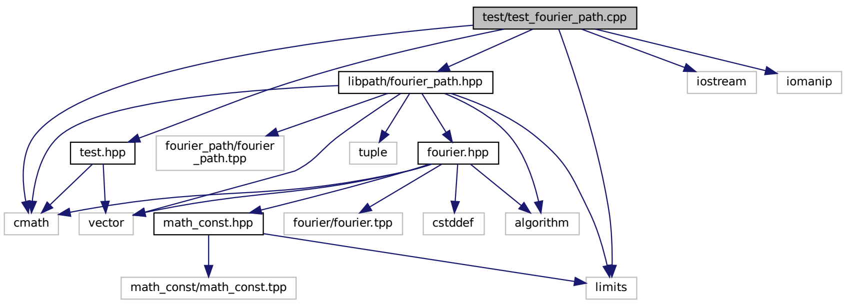
test result

15.20 test/test_fourier_path.cpp File Reference

test fourier_path class routine

```
#include <cmath>
#include <iostream>
#include <iomanip>
#include <limits>
#include "libpath/fourier_path.hpp"
#include "test.hpp"
```

Include dependency graph for test_fourier_path.cpp:



Functions

- int test::test_fourier_path ()

  test libpath::fourier_path class

15.20.1 Detailed Description

test fourier_path class routine

Author

pistack (Junho Lee)

Date

2021. 11. 12.

15.20.2 Function Documentation

15.20.2.1 test_fourier_path()

```
int test::test_fourier_path ( )
```

test libpath::fourier_path class

Returns

test result

Index