Google Summer of Code 2020 Project Proposal
GeomScale Project
Sampling from log-concave densities in high dimensions

Marios Papachristou
National Technical University of Athens
papachristoumarios@gmail.com

April 9, 2020

Abstract

This proposal aims to provide functionality on sampling from high-dimensional densities in the GeomScale project. A log-concave density is a density $g(x)$ proportional to $\exp(-f(x))$ where $f(x)$ is a (strongly) convex function subject to a convex polytope K. This proposal aims to implement the following samplers:

- Ball-walk (extension of existing functionality to log-concave densities)
- Hit-and-run (extension of existing functionality to log-concave densities)
- Hamiltonian Monte Carlo. [DKPR87, DQK+19]
- Randomized Midpoint Method for sampling using the Underdamped Langrevin Diffusion Process [SL19].

The samplers, such as Hamiltonian Monte Carlo, that require solving an ODE need functionality in order to do so. For that reason we will implement the following numerical ODE solvers for use with GeomScale:

- Euler Method
- Runge-Kutta-4 Method
- Collocation Methods
- Leapfrog Method

The back-end implementations will be developed in C++ and high-level functionality will be exposed to R through Rcpp.

Keywords. samplers, log-concave densities, boundary oracles, ode solvers, sde solvers

Contents

1 Contact Information **2**

2 Personal Statement **2**

3 Tests **7**
3.1 Easy Task: Simulation of Random Walks using GeomScale 7
3.2 Medium Task: Ball-Walk Sampling 7
3.3 Hard Task: Gradient Descent using the Barzilai-Borwein Method 8

4 Theoretical Background and Related Work **9**
4.1 Ball-Walk Sampling 10
4.2 Hit-and-Run Sampling 10
4.3 Hamiltonian Monte Carlo Sampling 11
4.3.1 Euler Method 12
4.3.2 Runge-Kutta-4 Method 12
4.3.3 Collocation Method 13
4.3.4 Leapfrog Method 14
4.3.5 HMC-ECS 15
4.4 Randomized Midpoint Method and the Underdamped Langrevin Diffusion . 16

5 Algorithm Implementations **18**
5.1 Oracles 18
5.2 Ball-Walk Sampling 18
5.3 Hit-and-Run Sampling 18
5.4 Hamiltonian Monte Carlo 19
5.5 Underdamped Langrevin Diffusion 19
5.6 Boundary Oracles 19
5.6.1 Motivation 19
5.6.2 Calculating Intersections with boundary ∂K 20
5.7 Boundary Reflections 23
5.8 Generalizations to non-linear General Convex Bodies 23

6 Proposal Outline **25**
6.1 Two-way integration with GeomScale 25
6.2 Related Frameworks 26
6.3 Milestones 27

7 Timeline **28**

8 Development Workflow **29**
8.1 Workflow 29
8.2 Mentors 29

9 Development Tools **29**
9.1 Deployment 29
9.2 Documentation 29
9.3 Testing 29

10 Indicative Applications of the Project **30**
10.1 Solvers for ODEs of the form $\dot{x}(t) = F(x, t)$ 30
10.2 Faster Inference for Latent Variable Models 30

11 Personal Commitments **32**

A Source Code of Tasks **34**
A.1 Easy 34
A.2 Medium 34
A.3 Hard 37

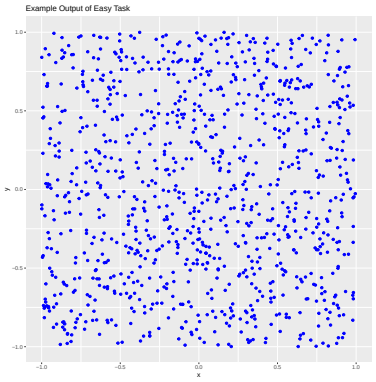


Figure 1: Results of Easy Task. We generated a 10-dimensional cube P and sampled $n = 1000$ points from the cube using default parameters.

3 Tests

In order to demonstrate interest on the GeomScale project, students had to solve some tests, which were dependent on the project. The tests included three challenges of varying levels of difficulty: easy, medium, and hard. The detailed list and solutions to the challenges will be outlined below as well as are uploaded to GitHub https://github.com/papachristoumarios/geomscales-challenge. Solutions to the problems were developed in the R language (with functionality from the C++ backend) and the C++ language using the Boost Library. Although much of the functionality is already included in GeomScale, we[2] reimplemented the Medium Challenge.

3.1 Easy Task: Simulation of Random Walks using GeomScale

In this task we were asked to examine how different random walk processes behave for sampling from a H-polytope K in $\mathbb{R}^d$. More specifically, we use GeomScale's builtin functionality to take samples using the random walks. The code of the easy challenge can be found in the Appendix.

Figure 1 shows the first two dimensions of the sampling procedure on a square.

3.2 Medium Task: Ball-Walk Sampling

In this task, we were asked to sample from a log-concave distribution $g(x)$, that is a distribution of the form $g(x) \propto \exp(-f(x))$ where $f(x)$ is a strongly convex function using the Ball-Walk Sampling Method. The Ball-Walk sampling method is similar to the Metropolis-Hastings Algorithm[3]. More specifically, the Ball-Walk Sampling Method gathers samples

[2]Refers to the academic "we" and corresponds to the author of the proposal.

[3]Also known as the "Metropolis Filter"

inside a polytope K with the following procedure starting from an initial position $x(0)$. That is for every $t \geq 1$

1. Get a sample y uniformly at random which is within distance δ from sample $x(t)$ and centered in $x(t)$. If the sample is outside the polytope then the next sample $x(t+1)$ equals the previous one.

2. Transit to the point with probability given by the Metropolis-Hastings algorithm, that is $p(x(t+1) \leftarrow y) = \min\{1, g(y)/g(x(t))\}$. Otherwise $x(t+1) \leftarrow x(t)$.

For a general function $h(x)$ we can approximate the quantity $\sum_{x \in K} h(x)g(x)$ — or in the continuous case the integral $\int_K g(x)h(x)dx$ — by calculating the ergodic mean of the above procedure, since from the Ergodic Theorem [Lou15], it holds

$$\lim_{T \to \infty} \Pr\left[\left|\sum_{x \in K} h(x)g(x) - \frac{1}{T}\sum_{t=1}^{T} h(x(t))\right| > \varepsilon\right] = 0 \tag{1}$$

The BW Sampler Source Code can be found in the Appendix.

3.3 Hard Task: Gradient Descent using the Barzilai-Borwein Method

In this task, we were asked to minimize a strongly convex function $f : A \to B$ using gradient descent and more specifically the Barzilai-Borwein (BB) method. The BB method has been outlined in [BB88] and can be summarized as follows. The typical process for Gradient Descent (GD) is the Steepest Descent rule

$$x(t+1) = x(t) - \eta(t)\nabla f(x(t)) \tag{2}$$

where $x(t)$ is the value of the solution (position) at step t, $\eta(t)$ is called the learning rate and $\nabla f(x(t))$ is the gradient of the function at step t. For example, a simple quadratic function $f(x) = \frac{1}{2}x^T Ax$ has an update rule of

$$x(t+1) = x(t) - \eta(t)Ax(t) \tag{3}$$

Although in its most simple form the function $\eta(t)$ is usually some constant value η for all t, an extended line of work has been outlined for this particular problem. For a concrete overview we redirect the interested reader to [SSBD14]. In our case, the BB method attempts to find a fast approximation to the Gauss-Newton update rule via solving a least squares problem. More specifically, the Gauss-Newton update rule attempts to minimize the function change towards a direction that minimizes the second-order Taylor expansion of the function, that is

$$\delta = \text{argmin}_{\delta \in A}\left\{f(x) + \delta^T \nabla f(x) + \frac{1}{2}\delta^T \nabla^2 f(x)\delta\right\} \tag{4}$$

where $\nabla^2 f(x)$ is the Hessian Matrix of f at point $x \in A$, where A is the domain of f. For a convex function, the expression attains a minimum when $\delta = (\nabla^2 f(x))^{-1}\nabla f(x)$, which yields the update rule

$$x(t+1) = x(t) - (\nabla^2 f(x))^{-1}\nabla f(x) \tag{5}$$

However, the computation of the inverse of the Hessian Matrix is usually computationally intensive and in [BB88], the authors suggested the following approximation: $\eta(t)\nabla f(x) \approx (\nabla^2 f(x))^{-1} f(x)$ with respect to the L_2 norm. The equivalent optimization problem is

$$\eta(t) = \text{argmin}_{\eta(t)} \frac{1}{2} \left\| \eta(t)\nabla f(x) - (\nabla^2 f(x))^{-1} f(x) \right\|_2^2 \tag{6}$$

As an example consider the quadratic function $f(x) = x^T Ax + b^T x$ where $A \succ 0$ and symmetric. The Hessian of the function is $\nabla^2 f(x) = A \succ 0$. We also define

$$g(t) = \nabla f(x(t)) \tag{7}$$
$$s(t-1) = x(t) - x(t-1) \tag{8}$$
$$y(t-1) = g(t) - g(t-1) \tag{9}$$

Using the equalities we can show that A satisfies $As(t-1) = y(t-1)$ hence our optimization problem reduces to computing

$$\eta(t) = \text{argmin}_{\eta(t)} \frac{1}{2} \left\| \eta(t)y(t-1) - s(k-1) \right\|_2^2 \tag{10}$$

Which has solutions

$$\eta_1(t) = \frac{s^T(t-1)s(t-1)}{s^T(t-1)y(t-1)} \tag{11}$$
$$\eta_2(t) = \frac{s^T(t-1)y(t-1)}{y^T(t-1)y(t-1)} \tag{12}$$

We can choose for $t \geq 1$ between the two learning rates uniformly at random and perform the minimization. The two solutions arise from solving the least squares problem as is in Eq. 10 or via dividing both sides by $\eta(t)$ and solving the alternate problem. The code of the hard challenge can be found in the Appendix.

4 Theoretical Background and Related Work

In this project we are asked to implement sampling methods from log-concave distributions. As noted before, a log-concave distribution $\mathcal{D}$ is a distribution with an associated probability density function (PDF) $g(x)$ that is defined as

$$g(x) = \frac{1}{Z} \exp(-f(x)) \tag{13}$$

where $f(x)$ is a strongly convex function[4]

The required oracles are the ones for the computation of $f(x)$ and $\nabla f(x)$[5] and the membership oracle $\mathbf{1}_K(x) = \mathbf{1}\{x \in K\}$ for answering whether x is in the polytope K[6]. Sampling can be unrestricted to the domain of $\mathcal{D}$ or to a convex polytope K. The sampling methods which are proposed are the following

1. Ball-Walk Sampling
2. Hit-and-Run
3. Hamiltonian Monte Carlo
4. Underdamped Langrevin Diffusion / Randomized Midpoint Method

The algorithms will be described below for completeness purposes.

4.1 Ball-Walk Sampling

The Ball-Walk (BW) sampling algorithm is described as part of the tasks needed for contacting the mentors of the GeomScale project. In a nutshell, given a sample x the sampler chooses a sample y that is within a distance $\delta > 0$ from x and lies in the polytope K. The sampler moves to y with probability $\min\{1, \exp(-f(y)+f(x))\}$. In the log-space it translates to $\min\{0, f(x) - f(y)\}$. The sampler starts from an initial point $x_0 \in K$. The algorithm is identical to the classical Metropolis-Hastings (or MCMC) algorithm with the only difference that the samples are taken from a continuous space and two consecutive samples are within distance δ from each other, subject to the polytope K.

4.2 Hit-and-Run Sampling

The Hit-and-Run (HR) sampler is another MCMC-inspired algorithm for generating samples from an arbitrary distribution $g(x)$ described in [Smi96]. The algorithm starts from an initial position $x_0 \in K$. Given that the sampler is at x, the sampler proceeds in choosing a random vector $d \sim \mathcal{D}$ and builds the line $y = x + \lambda d$ for $\lambda \in \mathbb{R}$. Then, it considers the set $L = K \cap \{y | y = x + \lambda d, \lambda \in \mathbb{R}\}$ and generates a point $y \in L$. The parameter λ that corresponds to this point has density $g_i(\lambda) = \frac{g(x+\lambda d)}{\int_K g(x+rd)dr}$.

[4] A function $f : A \to B$ is called m-strongly convex in the domain A if and only if for all $x, y \in A$ the following holds

$$(\nabla f(x) - \nabla f(y))^T (x - y) \geq m\|x - y\|_2^2$$

If the function is twice differentiable in A then the condition translates to the Linear Matrix Inequality (LMI) $\nabla^2 f(x) \geq mI$ for all $x \in A$.

[5] Provided externally.

[6] Provided in GeomScale.

4.3 Hamiltonian Monte Carlo Sampling

The Hamiltonian-Monte-Carlo (HMC) sampler (or Hybrid Monte Carlo) was introduced in [DKPR87]. According to this sampler we have a particle with position q and momentum p, which has a Kinetic Energy $\mathcal{K} = \mathcal{K}(p)$ and a potential energy $\mathcal{U} = U(q)$. The sum of the two energies is the mechanical energy of the system or the Hamiltonian $\mathcal{H}(p, q) = \mathcal{K}(p) + \mathcal{U}(q)$ such that

$$\dot{q} = \frac{\partial \mathcal{H}}{\partial p} \tag{14}$$

$$\dot{p} = -\frac{\partial \mathcal{H}}{\partial q} \tag{15}$$

The connection to MCMC is given as follows: The vector q refers to the parameters of the distribution $\mathcal{D}$ and the potential energy to the negative log-likelihood of $\mathcal{D}$. The momentum p is connected to the "augmented variables" such that the kinetic energy is calculated. The algorithm is repeated iteratively to calculate the q_i's such that $q_1, \ldots, q_n$ are samples from $\mathcal{D}$. Note that due to 14, the vectors p and q have the same dimensions, i.e. $p, q \in \mathbb{R}^d$. The exact procedure is described below

1. Sample $p \sim \mathcal{N}(0, I)$

2. $p_{start} \leftarrow p$

3. $q_{start} \leftarrow q_{i-1}$

4. q and p are generated from solving the ODEs of Eq. 14. We can use numerous methods to do it, such as Euler, Runge-Kutta-4, Collocation methods, Leapfrog integration and many more.

5. $p \leftarrow -p$

6. Advance to the next state with probability $\min\{1, \exp(\mathcal{H}(p, q) - \mathcal{H}(p_{start}, q_{start})\}$. Note that this condition for transition is needed for the classical HMC (Metropolized) defined in [DKPR87]. The state-of-the-art method of [LSV18] does not require a Metropolis filter and advances to the next state immediately.

Where the Kinetic Energy equals $p^T p$ and the potential energy equals $f(q)$. Another notation is v for the velocity and x for the position. The equations can be solved using numerical methods such as Euler's method, Runge-Kutta-4 method or collocation methods. Below we give out the iterative algorithms for a Hamiltonian function of the form $H(x, v) = \frac{1}{2} v^T v + f(x)$.

Concentration Bounds for Metropolized HMC. The Metropolized-HMC, that is the HMC that contains the Metropolis filter $\min\{1, \exp(\mathcal{H}(p,q) - \mathcal{H}(p_{start}, q_{start})\}$, has been recently shown in reference [LST20] to posses some very good properties when f is M-smooth and m-strongly convex that is, the norm of the gradient $\|\nabla f(x)\|$ is well-concentrated near $\mathbb{E}_{x\sim g}[\|\nabla f(x)\|]$, where $g(x)$ is the desired distribution. To be more specific, Lee et al. showed that

$$\Pr_{x\sim g}\left[\nabla f(x) \geq \mathbb{E}_{x\sim g}[\|\nabla f(x)\|] + c\sqrt{M}\log d\right] \leq 3d^{-c} \tag{16}$$

Moreover, they used the above result to state that the mixing time of the Markov Chain, using the Metropolized HMC algorithm with Leapfrog integration in order to sample from $g(x)$ is $\Omega(m/M)$ using a step size of $O(M^{-1/2})$.

Example: Gaussians. The distribution $g(x)$ is the density of the Normal distribution $\mathcal{N}(0, I)$ in d-dimensions. The function f equals to $f(x) = \frac{1}{2}x^T x$ with $\nabla f(x) = x$ and Hessian $\nabla^2 f(x) = I$. Using $H = \frac{1}{2}v^T v + \frac{1}{2}x^T x$ (system of a cart and a spring) we have that

$$\dot{x} = v \tag{17}$$
$$\dot{v} = -x \tag{18}$$

4.3.1 Euler Method

The classical Euler method for solving ODEs involves discretizing the ODE for T steps. We start from an initial point of time t_0 and compute the next point in time as a small displacement from the previous one, that is $t_{i+1} = t_i + h$. Now for updating the solution we make use of the differential

$$x_{i+1} = x_i + \left.\frac{\partial \mathcal{H}}{\partial v}\right|_{v=v_i} = x_i + v_i \tag{19}$$
$$v_{i+1} = v_i - \left.\frac{\partial \mathcal{H}}{\partial x}\right|_{x=x_i} = v_i - \nabla f(x_i) \tag{20}$$

for $0 \leq i \leq T$, where $x_i = x(t_i),\ v_i = v(t_i)$.

4.3.2 Runge-Kutta-4 Method

The Runge-Kutta-4 method (RK4) for an ODE of the form $\dot{y}(t) = F(y,t)$ and $y(t_0) = y_0$ involves a similar — but more sophisticated — procedure compared to the Euler method

where

$$t_{i+1} = t_i + h \tag{21}$$
$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{22}$$
$$k_1 = hF(y_i, t_i) \tag{23}$$
$$k_2 = hF(y_i + k_1/2, t_i + h/2) \tag{24}$$
$$k_3 = hF(y_i + k_2/2, t_i + h/2) \tag{25}$$
$$k_4 = hF(y_i + k_3, t_i + h) \tag{26}$$

Now we consider that the solution is the state vector $(x, v)^T$ and therefore we can write that $F = (v, -\nabla f)^T$. Therefore the RK4 equations become

$$t_{i+1} = t_i + h \tag{27}$$
$$k_1 = h \begin{pmatrix} v_i \\ -\nabla f(x_i) \end{pmatrix} \tag{28}$$
$$k_2 = h \begin{pmatrix} 3/2 v_i \\ -\nabla f(x_i + k_{1x}/2) \end{pmatrix} \tag{29}$$
$$k_3 = h \begin{pmatrix} 7/4 v_i \\ -\nabla f(x_i + k_{2x}/2) \end{pmatrix} \tag{30}$$
$$k_4 = h \begin{pmatrix} 11/4 v_i \\ -\nabla f(x_i + k_{3x}) \end{pmatrix} \tag{31}$$
$$\begin{pmatrix} x_{i+1} \\ v_{i+1} \end{pmatrix} = \begin{pmatrix} x_i \\ v_i \end{pmatrix} + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{32}$$

4.3.3 Collocation Method

The collocation method involves approximating the solution $y(t)$ of the ODE $\dot{y}(t) = f(y, t)$ with a polynomial of degree n, that is $p(t)$ such that

$$t_{i+1} = t_i + h \tag{33}$$
$$\dot{p}(t_{i+1}) = f(p(t_{i+1}), t_{i+1}) \tag{34}$$

The polynomial function $p(t) = \sum_{i=0}^{n-1} c_n t^n$ can be determined by solving the resulting system of equations which results in n individual conditions that have to be satisfied. For more information about the method, we redirect the interested reader to [AP98]. More recently, a collocation method was introduced in [LSV18] for solving ODEs.

The authors of [LSV18] derive a novel algorithm for solving multivariate ODEs whose solution is close to the span of a known basis of functions with nearly linear runtime with respect to the dimensionality. More specifically, the authors are mainly motivated from the logistic regression problem. In a nutshell, the loss of the logistic regression is

$$\sum_i \phi_i(a_i^T x) \tag{35}$$

where a_i's are the weights, $\phi(t) = 1/(1+\exp(-t))$ is the logistic sigmoid function and x is the feature vector which is directly related to log-concave sampling. More specifically, the authors refer to the Underdamped Langrevin Diffusion process (ULD) Stochastic Differential Equation (SDE) as a method for sampling from log-concave densities. The ULD process is further described in Section 4.4 where the Randomized Midpoint Method is introduced.

As for the general collocation method, the current paper brings down the total time needed for solving the ODE in $\kappa^{1.5}d$ where d is the number of variables of $f(x)$ and κ is the condition number of $\nabla^2 f(x)$[7] . Moreover, the authors describe a collocation method for solving such ODEs of the form $\dot{x} = F(x,t),\ x(0) = v$ which is based on the idea that $\dot{x}(t)$ can be approximated by a linear combination of basis functions $\phi_j(t)$. By Gram-Schmidt orthogonalization we know that there exist c_j for which

$$\dot{x}(t) \approx \sum_{j=1}^{D} \dot{x}(t = c_j)\phi_j(t) = \sum_{j=1}^{D} F(x(c_j), c_j)\phi_j(t)$$

for which reason the set of equations described in Eq. **??** can yield a solution of form

$$x(t) \approx v + \int_0^t \sum_{j=1}^{D} F(x(c_j), c_j)\phi_j(t) \tag{36}$$

This computation can be done a la fixed point — namely by applying an integral operator multiple times. The quantity inside the integral is represented by a matrix-matrix product that is $F(X,c)_{i,j} = F(X_{*,j}, c_j)_i$ and A_ϕ defined by $(A_\phi)_{i,j} = \int_0^{c_j} \phi_i(s)ds$

The matrix A_ϕ is constant througout every iteration and is the matrix of basis integrand functions. Intuitively the entries of the matrix look like the classical unitary transform matrices (like DFT for example). Then, the authors define the operator $\mathcal{T}(X) = v1_D^T + F(X,c)A_\phi$ and apply it iteratively for $N = \left\lceil \log\left(\frac{T}{\epsilon}\max_{s\in[0,T]} \|F(v,s)\|\right)\right\rceil$ steps. The operator is a contractive mapping[8] since the operator $F(X,c)$ is contractive. This collocation algorithm is used in order to generate samples for the HMC procedure, in the absence of a Metropolis filter.

4.3.4 Leapfrog Method

Additionally, for purposes of completeness, we refer to the Leapfrog method in order to solve a second order differential equation of the form $\ddot{x}(t) = F(x)$. The Leapfrog method transforms the previous equation to

[7]For a twice differentiable function f that is M-smooth and m-strongly convex, that is $mI \succeq \nabla^2 f(x) \succeq MI$ the condition number κ is the ratio of the maximum to the maximum eigenvalue to the minimum eigenvalue that is $\kappa = m/M$.

[8]A mapping $f : X \to Y$ between two metric spaces (X, d_X) and (Y, d_Y) is contractive if and only if there exists $L \in (0,1)$ such that forall $x, y \in X$ the inequality $d_Y(f(y), f(x)) \leq L d_X(y,x)$ holds. All contractive mappings have a fixed point due to the Banach fixed point theorem. Banach's theorem also states that someone can start from a point $x_0 \in X$ and apply the mapping multiple times as $x_{n+1} = f(x_n)$ until he reaches the fixed point.

$$\dot{x} = v \tag{37}$$
$$\dot{v} = F(x) \tag{38}$$

The Leapfrog method advances as follows

$$v_i' = v_i + \frac{\eta}{2} F(x_i) \tag{39}$$
$$x_i' = x_i + \eta v_i' \tag{40}$$
$$v_i = v_i' + \frac{\eta}{2} F(x_i') \tag{41}$$

In the case of the HMC algorithm the function $F(x)$ equals to $-\nabla f(x)$, hence the equations become

$$v_i' = v_i - \frac{\eta}{2} \nabla f(x_i) \tag{42}$$
$$x_i' = x_i + \eta v_i' \tag{43}$$
$$v_i = v_i' - \frac{\eta}{2} \nabla f(x_i') \tag{44}$$

The quantity η is callsed the learning rate, like in the case of gradient descent.

4.3.5 HMC-ECS

Here, we describe the method of [DQK+19] for HMC with *subsampling*. In this setting, samples from a distribution $g(x)$ need to be obtained. A computationally attractive way to accelerate HMC is to use a fixed subsample of the data to unbiasedly estimate the computationally costly gradients in each step of the discretized Hamiltonian trajectory, and skip the MH correction to avoid evaluating the posterior on the full data. In literature, the pseudo-marginal MCMC methods introduce m auxiliary variables $u \in \mathbb{R}^m$ distributed with density $p_U(u)$ and target the posterior distribution

$$\bar{g}_m(x, u) \propto \hat{L}_m(x) g(x) p_U(u) \tag{45}$$

where $\hat{L}_m(x)$ is an unbiased and non-negative estimator of the likelihood $L(x)$ based on the auxiliary variables u. Since the estimator is unbiased, generating samples for $\bar{g}_m$ means generating samples for $g(x) = \exp(-f(x))$. The authors introduce the HMC philosophy into the algorithm and tweak Eq. 45 to

$$\bar{g}_m(x, u) \propto \exp(-\mathcal{H}(x, v)) p_U(u) \tag{46}$$
$$\mathcal{H}(x, v|u) = \mathcal{K}(v) + \mathcal{U}(x|u) \tag{47}$$
$$\mathcal{K}(v) = \frac{1}{2}\|v\|^2 \tag{48}$$
$$\mathcal{U}(x) = f(x) - \log \hat{L}_m(x|u) \tag{49}$$

The solution to the Hamiltonian Equations is done via the Leapfrog method, described in Section 4.3.4. At iteration $j-1$, a candidate sample $u \sim p_U$ is selected and the transition from $u^{(j-1)}$ to u is done with

$$a_u = \min\left\{1, \frac{\hat{L}_m(x|u)}{\hat{L}_m(x|u^{(j-1)})}\right\} \tag{50}$$

Then we proceed with applying the classical HMC algorithm (solve the Hamiltonian equations and transist with a Metropolis probability dependent on the starting and ending Hamiltonian values. The paper proposes variational approximation of the log-likelihood $\ell(x) = \log L(x)$ using variational methods (control variables). The estimate $\hat{\ell}_m(x)$ of the log-likelihood is then used for estimating the likelihood $\hat{L}_m(x)$. More specifically, the authors propose the estimator to be derived from sampling m observations $u_1, \ldots, u_m \sim p_U$ with replacement

$$\begin{aligned}
\ell_{u_i}(x) &= \log \Pr[u_i|x] \\
\hat{\ell}_m(x) &= \sum_{i=1}^{n} q_{u_i}(x) + \hat{d}_m(x) \\
\hat{d}_m(x) &= \frac{1}{nm}\sum_{i=1}^{m} d_{u_i}(x) \\
d_{u_i}(x) &= \ell_{u_i}(x) - q_{u_i}(x) \\
\hat{\sigma}_m^2(x) &= \frac{n^2}{m^2}\sum_{i=1}^{m}(d_{u_i}(x) - \bar{d}_{u_i}(x))^2 \\
q_{u_i}(x) &= \ell_{u_i}(\theta_0) + \nabla\ell_{u_i}(\theta_0)^T(\theta-\theta_0) + \frac{1}{2}(\theta-\theta_0)^T\nabla^2\ell_{u_i}(\theta_0)(\theta-\theta_0) \\
\hat{L}_m(x) &= \exp\left(\hat{\ell}_m(x) - \frac{1}{2}\hat{\sigma}_m^2(x)\right)
\end{aligned}$$

This approach to subsampling in HMC estimates both the Hamiltonian and its corresponding dynamics from the same subsample.

4.4 Randomized Midpoint Method and the Underdamped Langrevin Diffusion

Perhaps, the most challenging part of the project is to implement the Randomized Midpoint Method (RMM) introduced in reference [SL19]. The RMP method is a state-of-the-art (SOTA) method for sampling from log-concave distributions $g(x) \propto \exp(-f(x))$ where $f : K \to \mathbb{R}$ is m-strongly convex and ∇f is L-Lipschitz in the domain K. More specifically, the authors claim a significant performance gain for learning the desired distribution $g(x)$ in the statistical sense (W_2-distance). The authors propose a method for discretizing Stochastic Differential Equations (SDEs) of the form $\dot{x} = F(x,t)$. The method that is followed

transforms the SDE into the integral equation

$$x(t) = x(0) + \int_0^t F(x,s)ds = \mathcal{T}(x) \tag{51}$$

Where the fixed point $x(t)$ is approximated by sequentially applying the integral operator $\mathcal{T}$. The convergence rate of the above iterative procedure is shown to be $O(\bar{\kappa}^{3/2})$ for $\bar{\kappa} := L/m$, where L is the Lipschitz constant and m is the constant that determines the strong-convexity of f. Moreover, the authors devise a method for approximating

$$\int_0^t F(x,s)ds \sim \sum_{t=1}^{T} w_t F(x,s_t) \tag{52}$$

The RMM algorithm — which is an MCMC-related method — is introduced to handle this integration. More specifically, the RMM algorithm resides in sampling using the Underdamped Langrevin Diffusion Stochastic Process

$$dx(t) = -\nabla f(x(t)) + \sqrt{2}dB_t \tag{53}$$

where $\{B_t\}_{1\leq t\leq T}$ is the Brownian Motion Stochastic Process. The ULD process can be explained by an HMC approach via considering the system of equations

$$dv(t) = -2v(t)dt - u\nabla f(x(t))dt + 2\sqrt{u}dB_t \qquad dx(t) = v(t)dt \tag{54}$$

Where $u := 1/L$. The randomized midpoint method proceeds in N steps where in step $1 \leq n \leq N$, a number a is chosen from $U[0,1]$ and then the real d-dimensional vectors $W_1^{(n)}, W_2^{(n)}, W_3^{(n)}$ are chosen such that

$$W_1 = \int_0^{ah} (1 - \exp(-2(ah-s))dB_s \tag{55}$$

$$W_2 = \int_0^{h} (1 - \exp(-2(h-s))dB_s \tag{56}$$

$$W_3 = \int_0^{h} \exp(-2(h-s))dB_s \tag{57}$$

where the terms W_1, W_2, W_3 can be further approximated as devised in [SL19]. Then the values are updated as

$$x_{n+1/2} = x_n + \frac{1}{2}(1-\exp(-2ah))v_n - \frac{1}{2}u(ah - \frac{1}{2}(1-\exp(-2ah)))\nabla f(x_n) + \sqrt{u}W_1^{(n)} \tag{58}$$

$$x_{n+1} = x_n + \frac{1}{2}(1-\exp(-2h))v_n - \frac{1}{2}uh(1 - \frac{1}{2}\exp(-2ah))\nabla f(x_{n+1/2}) + \sqrt{u}W_2^{(n)} \tag{59}$$

$$v_{n+1} = v_n \exp(-2h) - uh\exp(-2(h-ah))\nabla f(x_{n+1/2}) + 2\sqrt{u}W_3^{(n)} \tag{60}$$

The authors also extend this method such that the solution can be computed in parallel. We redirect the interested reader to the respective paper. The collection of samples

(x_1, v_1) through (x_N, v_N) are the solution to the Underdamped Langrevin Diffusion Differential Equation and moreover the x_i's are "close" to samples of the ideal distribution $g(x)$ in the 2-Wasserstein Distance where the p-Wasserstein Distance is given as

$$W_p(\mu, \nu) = \left(\inf_{(X,Y) \in \mathcal{C}(\mu,\nu)} \mathbb{E}\left[\|X - Y\|^p \right] \right)^{1/p} \tag{61}$$

where μ, ν are probability measures defined on Ω_1 and Ω_2 respectively and $\mathcal{C}(\mu, \nu)$ is the set of all their couplings.

5 Algorithm Implementations

Below we provide the essential technical parts — from a technical perspective — that need to be implemented for the project proposal.

5.1 Oracles

The three oracles which responsible for returning $f(x)$ and $\nabla f(x)$ can be either implemented in C++ or the R language as external routines. The oracle computation external functionality will be included to the library main functionality and provide functions for the three oracles. Furthermore, the oracle $\mathbf{1}_K(x)$ is already implemented in GeomScale for convex bodies. A separate section (Section 5.6) is devoted to algorithmic issues which arise from the intersection of curves with convex bodies (polytopes and general non-linear convex bodies).

5.2 Ball-Walk Sampling

For the BW sampler for picking a sample y that's within distance $\delta > 0$ from another sample x, a direction z can be chosen uniformly at random in the ball $B(0, \delta)$ such that $\|z\| \leq \delta$ and $x + z \in K$ and y can be set as $y := x + z$. We can use the Müller method to sample from the high-dimensional ball

- Choose $u \sim \mathcal{N}(0, I)$ (in d-dimensions)
- Choose $r \sim U[0, \delta/d]$
- Return $x = ru/\|u\|$

The BW sampler is already implemented in GeomScale on the `samplers/samplers.h` file for uniform distributions and the spherical normal distribution. Our proposal will extend it to log-concave distributions using the evaluation oracle for $f(x)$.

5.3 Hit-and-Run Sampling

The Hit-and-Run along with most of the functionality (intersections of lines with convex polytopes) are already implemented in GeomScale for uniform sampling. Our proposal will extend the HR sampler to log-concave distributions via incorporating the Metropolis filter

(if the function f is only known, computation can be done using log-exp-sum-trick in order to avoid numerical overflows).

5.4 Hamiltonian Monte Carlo

In this algorithm the exact description of the numerical methods suffices to solve the ODE via one of the three proposed methods. The HMC sampler is not implemented in GeomScale. Our proposal implement the HMC sampler from general log-concave distributions. Solutions to general ODEs, via the three main methods (Euler, RK4 and collocation) will be developed externally as a library (.h file in the case of GeomScale) in order to be able to be used independently.

5.5 Underdamped Langrevin Diffusion

The ULD sampler, which can be seen as a derivative of the HMC sampler together with a collocation method using exponential functions is not currently implemented in GeomScale. First of all, we will develop functionality for generating Brownian Motion Processes B_t. More specifically each sample B_t in a Brownian Process can be derived from B_s for $s \leq t$ via adding a normal variable of the form $\mathcal{N}(0, t-s)$. Then the next move is to calculate integrals of the form

$$\int_l^u f(s|h)dB_s \tag{62}$$

In order to calculate $W_1^{(n)}, W_2^{(n)}, W_3^{(n)}$ for the RMM method. The integrals can be approximated numerically via dividing the $[l, u]$ interval with step τ and taking $K = \lceil \frac{u-l}{\tau} \rceil$ Gaussian samples $X_1, \ldots, X_K$ that are distributed according to $\mathcal{N}(0, \tau)$ and then computing the Riemmannian sum $\sum_{k=1}^{K} f(X_k|h)X_k\tau$. Intervals can — of course — be of non-equal length in case faster computation is deserved. After that, the iterative procedure can be followed, as described in the research paper.

5.6 Boundary Oracles

5.6.1 Motivation

In multiple cases, we are requested to sample inside a convex polytope K. For example, in the HR sampler we draw a direction d from the distribution and then form the line $y = x+\lambda d$ and attempt to find where this line intersects the convex polytope K. For this reason we need efficient algorithms to perform such tasks.

To set things formally, we will assume that the convex polytope K can be represented in two ways

- The H-representation: The convex polytope K is defined by the system of inequalities (halfspaces) $Ax \leq b$.
- The V-representation: The convex polytope K is defined by its convex hull $CH(K) = \{v_1, \ldots, v_n\}$.

5.6.2 Calculating Intersections with boundary ∂K

V-representation. Now, we study and generalize the main algorithms used to find the intersection of the convex polytope K with the generalized curve

$$p(t) = \sum_{j=1}^{m} \phi_j(t) p_j \tag{63}$$

We will first see how the problem is solved when $p(t)$ is a line segment, that is

$$p(t) = tp_0 + (1-t)p_1 \qquad t \in [0,1] \tag{64}$$

Although at first glance one can use binary search and the oracle to find the t for which the line meets with the polytope, this solution is not efficient in terms of computational complexity. Instead, we need to use a faster method for the task. A classical algorithm to perform this kind of operation is the Cyrus-Beck algorithm [CB78]. Given the V-representation of a polytope K. Let $e_i = v_{(i+1) \mod n} - v_{i \mod n}$ be the edge vectors which join the vertices $v_{(i+1) \mod n}$ and $v_{i \mod n}$ together and let n_i be the outward normal vectors to e_i — that is $n_i^T e_i = 0$ for all i — and $p(t) = tp_0 + (1-t)p_1$ be a line segment parametrized by $t \in [0,1]$ with endpoints p_0, p_1. Then for every point $u \in \partial K$ we identify the following cases for the quantity $n_i^T(p(t_i) - u)$

- If the dot product is positive then $p(t_i) - u$ is interior to the convex polytope K.
- If the dot product is negative then $p(t_i) - u$ is exterior to the convex polytope K.
- If the value is equal to 0 then $p(t_i) \in \partial K$.

More analytically, if we expand the dot product we get

$$n_i^T(tp_0 + (1-t)p_1 - u) = 0 \iff n_i^T(p_0 - u) + n_i^T(p_1 - p_0)t_i = 0 \iff t_i = -\frac{n_i^T(p_0 - u)}{n_i^T(p_1 - p_0)} = \frac{N_i}{D_i} \tag{65}$$

If $t_i \notin [0,1]$ then we ignore the point. Otherwise we keep $t_u = \inf_i\{t_i \in [0,1] | D_i < 0\}$ and $t_l = \sup_i\{t_i \in [0,1] | D_i > 0\}$. The intersection points $p_l, p_u \in \partial K$ can be found by substituting $t = t_l$ anr $t = t_u$ respectively, that is $p_u = p(t = t_u)$ and $p_l = p(t = t_l)$.

Finally, the more general form of a curve $p(t) = \sum_{j=1}^{m} \phi_j(t) p_j$ — for example take the case when $\phi_j(t) = t^{j-1}$ — needs to be investigated we can reapply the more general CB algorithm to arrive at the equation

$$n_i^T \left(\sum_{j=1}^{m} \phi_j(t) p_j \right) = n_i^T u_i \iff \sum_{j=1}^{m} (n_i^T p_j)\phi_j(t) - n_i^T u_i = 0 \tag{66}$$

Now in order to find t we can use a standard numerical method, such as Newton's (or Newton-Raphson) Method[9]. More specifically, we can find t — if real roots exist — via the

[9] Theoretically, it is not the fastest method. Fast real root finders for polynomials have been laid out by Pan [Pan15]

iterative procedure

$$t^{(r+1)} = t^{(r)} - \frac{\sum_{j=1}^{m}(n_i^T p_j)\phi_j(t = t^{(i)}) - n_i^T u_i}{\sum_{j=1}^{m}(n_i^T p_j)\phi_j'(t = t^{(i)})} \tag{67}$$

For example in the case of the polynomial functions $\phi_j(t) = t^{j-1}$

$$t^{(r+1)} = t^{(r)} - \frac{\sum_{j=1}^{m}(n_i^T p_j)(t^{(i)})^{j-1} - n_i^T u_i}{\sum_{j=2}^{m}(n_i^T p_j)(j-1)(t^{(i)})^{j-2}} \tag{68}$$

An illustration of the problem is given at Figure 2 between a convex region and a parabola. Note that a polynomial of degree n can have at most $2n$ intersections with the convex region. Finally, in case we are at a point $p_0 = p(t = t_0)$ and we are interested in finding the "correct" root, that is the root near p_0 that intersects the polytope K, we have to start the Newton iteration from p_0, that is set $t^{(0)} = t_0$. Intuitively, if we have a "good-behaving" function then the Newton iteration at step $r + 1$ will be closed to the desired root, compared to step r.

H-representation. Finally, if we are given the H-representation (dual to the V-representation) of the polytope K parametrized by matrices A and b, then we can write the curve $p(t)$ as $X^T\Phi(t)$ where X contains the row vectors p_j^T and $\Phi(t)$ is a row vector containing the basis functions. Then for every t such that $p(t) \in K$ we have that

$$Ap(t) \leq b \iff (AX^T)\Phi(t) \leq b \tag{69}$$

We now define the matrix $\tilde{A} = AX^T$ such that

$$\tilde{A}\Phi(t) \leq b \tag{70}$$

On the intersection with the polytope K some constraints become tight, that is there is a subset of indices S such that

$$\tilde{A}_S\Phi(t) = b_S \qquad \tilde{A}_{\bar{S}}\Phi(t) < b_{\bar{S}} \tag{71}$$

where $\bar{S}$ denotes the complementary indices. Then we can use conventional root-finding — such as NR as above — in order to find the intervals $I_1, \ldots, I_M$ at which each line of the inequalities holds[10].For simplicity, assume that all inequalities involve continuous functions on domains $D_1, \ldots, D_M \subseteq \mathbb{R}$. Then by the Intermediate Value Theorem, we know that the corresponding functions retain their sign between the roots, so it suffices to calculate the functions at values between the roots (one per interval) to fin the desired intervals $I_1, \ldots, I_M$. Then we can take $I = \bigcap_i I_j = \bigcup_k [t_{lk}, t_{uk}]$ where the intersection can be computed with the following algorithm

1. Suppose that each I_i can be expressed as a union of contiguous intervals
2. Sort all the individual contiguous intervals by starting times (e.g. with quicksort). Initialize a min-heap.

[10]Assume there are M linear inequalities

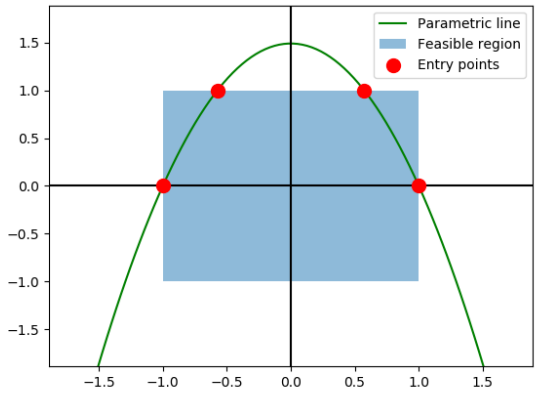


Figure 2: Illustration of the problem for intersection of the convex region $|x_1| \leq 1.5, |x_2| \leq 1$ and the parabola $p(t) = (0, 1.5) + (1, 0)t + (-1.5, 0)t^2$. The parabola meets the convex polygon to 4 points $(-1, 0), (1, 0), (-0.5, 1), (0.5, 1)$.

3. Iterate from left-to-right. If the current starting point violates the min of the min-heap pop the interval on the top of the heap and repeat until the top of the heap is greater than the current start. Add the ending point to the min-heap.

4. In case you complete one interval from each of the halfspaces, find the intersection (pointwise max of starts and pointwise min of ends) and output the solution, if one root suffices. Otherwise, continue by keeping this result to a list to return it later.

5. If the algorithm terminates without solution return no solution.

Computational Complexity. Suppose that we have a d-dimensional convex polytope K which is represented by M halfspaces or by N vertices in the convex hull and the curve $p(t)$ is parametrized by m terms. The original Cyrus Beck algorithm has worst-case complexity of $O(Nd)$ since we iterate over the $n-1$ normals and have operations of cost of $O(d)$. For the more general method (worst-case), we need to calculate the $O(Nmd)$ values of $n_i^T p_j$ and then the complexity of NR has to be taken into account. However, note that using a polynomial which may have more real roots as the degree increases, so we expect some trade-off (amortized). Finally, in the algorithm for the H-representation, if there's a total of K intervals and we have M halfspaces then the total worst-case complexity is $O(K \log K + KM \log M)$ for finding all the intervals $[t_{lk}, t_{uk}]$. Of course a choice has to be made for choosing either the H-representation or the V-representation in case one has an exponential representation size and the other has a polynomial representation size (and vice versa). Classical examples are the L_1 ball $\|x\|_1 \leq 1$ and the L_∞ ball $\|x\|_\infty \leq 1$. The L_1 ball has an exponential H-representation (with respect to the dimension) and a polynomial V-representation and the L_∞ ball has an exponential V-representation and a polynomial H-representation.

5.7 Boundary Reflections

When the distribution $g(x)$ is truncated to a convex polytope K, when using a numerical method, one wants the solution $x(t)$ to lie inside the polytope. That is on the boundary ∂K the differential vector $dx(t) = \dot{x}(t)dt$ has to be perpendicular to the current surface normal n, that is

$$\frac{\partial x(t)}{\partial n} = n^T \dot{x}(t) = n^T F(x(t), t) = 0 \qquad \forall t \geq 0 \text{ such that } x(t) \in \partial K \tag{72}$$

With the term "current" we mean the normal (constraint) at which the solution will violate under an infinitesmall evolution in the time domain. Therefore we need to reflect about the boundary normal. These conditions are well-known in the domain of ODEs/PDEs as the Neumann condition. Of course, these are not the only boundary conditions for solving an ODE/PDE. For instance, there are the Dirichlet conditions which are imposed to the solution $x(t)$ itself, the Cauchy conditions where the boundary constraints are mixed and the Robin conditions where $ax(t) + b\frac{\partial x(t)}{\partial n} = g$ on the boundary ∂K. For purely algorithmic reasons the Neumann reflective conditions (the normal derivative equals zero) are faster than the other types of conditions. Suppose that we have a ray between p_0 which lies inside K and p_1 that is on ∂K[11]. Let $v = p_1 - p_0$ be the ray vector. Suppose also that the outward unit surface normal of the hyperplane that the ray meets with the polytope is n. Then the reflection we seek conserves the falling angle $-n^T v$ to the angle $-n^T v'$ where $v' = p_1 - p_0'$ is the reflected ray. Now, by simple geometric computations we can show that

$$p_0' = -2(v^T n)n - p_1 + 2p_0 \tag{73}$$

Therefore computation of the reflection can be done exactly in $O(d)$. Finally, boundary reflections are already implemented in GeomScale.

5.8 Generalizations to non-linear General Convex Bodies

[12]

In general, a boundary oracle can generalize to a non-linear convex body. The examples of general non-linear convex bodies are ample. The simplest one is the d-dimensional ball centered in x_0 with radius δ, that is

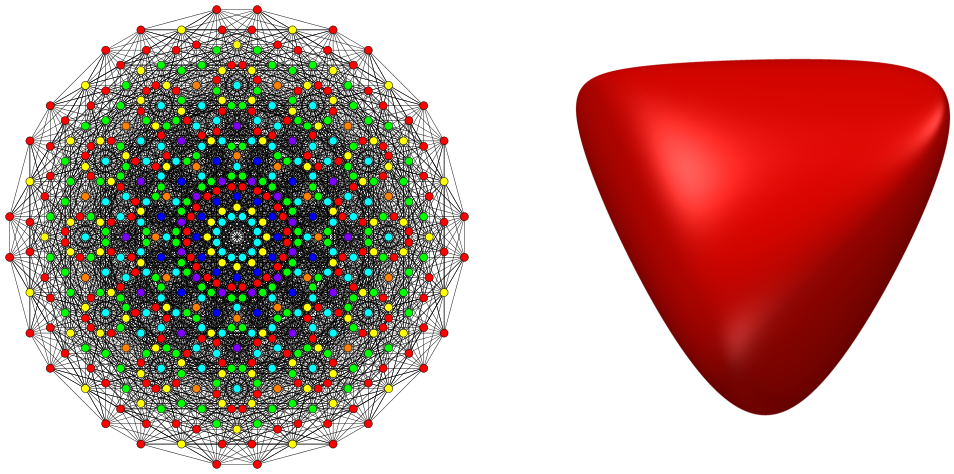
$$B(x_0, \delta) = \{x \in \mathbb{R}^d | \|x - x_0\| \leq \delta\}$$

or the cylinder of height h and radius δ in $\mathbb{R}^3$, modelled by

$$x_1^2 + x_2^2 \leq \delta^2 \qquad \text{and} \qquad -h/2 \leq x_3 \leq h/2$$

Convex bodies are fundamental forms in convex optimization. For example, polytopes arise from the intersection of halfspaces and are domains of problems in Linear Programming. Spectrahedra, which are convex bodies of the form $A(x) \succeq 0$ where

[11] If we do not know p_1 we can use the algorithms of the previous subsection to do so.
[12] Beyond GSoC 2020.



(a) Polytope (Domain of LP problems)

(b) Spectrahedron (Domain of SDP problems)

Figure 3: Examples of convex bodies.

$$A(x) = A_1 x_1 + A_2 x_2 \cdots + A_n x_n \qquad A_i = A_i^T \succeq 0$$

serve as domains for Semi-definite Programming. Figure 3 depicts two examples: a polytope and a spectrahedron. Generally speaking, we shall model a non-linear general convex body K with a set of M functions $f_1(x), \ldots, f_M(x) : A \to \mathbb{R}$ where $A \subseteq \mathbb{R}^d$ which satisfy the inequality $f_i(x) \leq 0$ for all $1 \leq i \leq M$. For example, the d-dimensional ball is modelled by $f(x) = \|x - x_0\|^2 - \delta^2$. Now, the set K is defined by

$$K = \{x \in A |\ f_i(x) \leq 0,\ 1 \leq i \leq M\} \tag{74}$$

Note that for the set K to be convex we require that for all $x, y \in K$ the quantity $tx + (1-t)y \in K$ for all $t \in [0,1]$. Indeed, if the functions f_i are convex then by Jensen's Inequality

$$f_i(tx + (1-t)y) \leq t f_i(x) + (1-t) f_i(y) \leq 0 \qquad 1 \leq i \leq M \tag{75}$$

For now, assume that all f_is are convex and that the sub-gradients $\nabla f_i(x)$ are defined in the domain A (we refer to the subgradients as the generalization of functions that are $C^k(A)$ for $k \geq 1$). Let $x_0 \in K$ be a point in K. Then by Taylor's Theorem near the point x_0[13]

$$f_i(x) \approx f_i(x_0) + \nabla f_i^T(x_0)(x - x_0) \tag{76}$$

We also define the function $F : A \to \mathbb{R}^M$ with $F(x) = (f_1(x), \ldots, f_M(x))$ and the Jacobian Matrix $J_F(x)$ with entries $(\nabla f_i^T(x))_j$ of the subgradients of the functions at x_i.

[13]If the Hessian oracles are available one can take the 2nd order approximation.

Now, the constraints $f_i(x) \leq 0$ near a point x_0 can be translated as

$$\nabla f_i^T(x_0)x \leq -f(x_0) + \nabla f_i^T(x_0)x_0 \qquad 1 \leq i \leq M \tag{77}$$

More formally, using F we can write

$$J_F(x_0)x \leq -F(x_0) + J_F(x_0)x_0 \tag{78}$$

We now recognize the familiar form $Dx \leq b$ of the convex polytope $P(K, x_0) = \{x \in A | J_F(x_0)x \leq -F(x_0) + J_F(x_0)x_0\}$ that is the *convex polytope approximation to the non-linear body K at point $x_0 \in K$* in the H-representation. The V-representation can be also obtained by applying the Beneath-Beyond algorithm. Now, we can use the previous methods for convex polytopes to find the points of intersection.

In the case where $p(t)$ is a curve as described above and $p_0 = p(t = t_0)$ is a point in K, we use the following algorithm to find an (approximate) intersection point of $p(t)$ with K

1. Start from p_0 and define the convex polytope approximation $P(K, p_0)$ near p_0.

2. Use a routine, e.g. for the H-representation of $P(K, p_0)$, to find an approximate point of intersection of $p_1 = p(t = t_1)$ and $P(K, p_0)$. If the functions f_i are two times differentiable with $mI \succeq \nabla f_i^2$, the first order approximation errors R_i satisfy

$$R_i = \sup_{x \in A} \|f(x) - f(x_0)\| \leq \frac{m\|p_1 - p_0\|^2}{2} \leq \frac{m \times \text{diam}^2(K)}{2}$$

   per coordinate.

3. We move at p_1 and approximate with the polytope $P(K, p_1)$. We now find a new point of intersection $p_2 = p(t = t_2)$. If $t_2 = t_1$ or $\|p_2 - p_1\| > \|p_1 - p_0\|$ we stop. Otherwise, if $t_2 \neq t_1$ or $\|p_2 - p_1\| \leq \|p_1 - p_0\|$ we continue. We can either iterate for T times and return p_T or until the relative error becomes less than some $\varepsilon > 0$.

6 Proposal Outline

6.1 Two-way integration with GeomScale

The purpose of this sub-section is to describe how our contribution will be integrated in GeomScale, namely which functionality from the existing GeomScale codebase it will utilize, as well as provide an indicative plan of how the new modules will be introduced. The main philosophy of GeomScale follows a procedural programming approach for most of its active functionality, with the exception of the definitions of the various objects (convex bodies, points etc.) of geometrical interest. The current proposal aims to build on this philosophy in the following ways.

- **ODE Solvers:** The proposed ODE solvers will be implemented in a separate package under `include/ode_solve.h`. More specifically we will provide API functionality of the form `ode_euler(...)`, `ode_rk4(...)`, `ode_collocation(...)`. The solver will be provided with the initial point x_0 upon which the ODE should evolve through the `point` class and the model $F(x, t)$ through `std::function` or as a function pointer (as in pure C). Moreover, additional parameters such as the tolerance for checking whether the solution has converged can be provided. In case an object-oriented approach is desired, we propose the definition of the abstract mother class `GenericODESolver`, from which `EulerODESolver`, `RK4ODESolver` and `CollocationODESolver` will inherit. The classes will act as *functors*, that is `operator ()` will be overloaded with the appropriate functionality.
- **Samplers:** The proposed Samplers will follow the same philosophy as the ODE solvers described above. To be more specific, the BW and HR sampler will be extended to support arbitrary (log-concave or general) distributions. The HMC and RMM samplers will be implemented in a similar way. Some example functionality of the existing GeomScale API is `line_intersect` and `line_intersect_coord`.
- **Porting to R:** The porting procedure to the R language will be done through the Rcpp[14] package. The wrappers for the R language will follow the same structure as the existing ones under `R-proj/src`. Error handling will be done in the R level through `Rcpp::exception`.

6.2 Related Frameworks

In this section, we will refer to two related frameworks that implement the HMC samplers. The two main frameworks are TensorFlow[15] and mc-stan[16]. First of all, the API call for HMC in the TensorFlow library is

```
tfp.mcmc.HamiltonianMonteCarlo(
target_log_prob_fn,
step_size,
num_leapfrog_steps,
state_gradients_are_stopped=False,
step_size_update_fn=None,
seed=None,
store_parameters_in_results=False,
name=None
)
```

and on mc-stan the ODE sovler specification is, for example for RK4

```
real[ , ] integrate_ode_rk45(function ode, real[]
   initial_state, real initial_time, real[] times, real[]
   theta, real[] x_r, int[] x_i)
```

[14] https://cran.r-project.org/web/packages/Rcpp/index.html
[15] https://www.tensorflow.org/probability/api_docs/python/tfp/mcmc/HamiltonianMonteCarlo
[16] https://mc-stan.org/docs/2_19/functions-reference/functions-ode-solver.html

We observe that the `HamiltonianMontecarlo` function call takes the following parameters

- The target log-propability function $-f(x)$
- The step size η for Leapfrog integration
- The number of total steps T
- The step size update rule in case we want a variable learning rage $\eta(t)$
- The random seed

and the ODE solver contains

- The target function $F(x,t)$ such that $\dot{x} = F(x,t)$
- The initial condition $x_0 = x(t = t_0)$
- The initial time t_0
- The parameter values θ
- The data to evaluate the solver x_r and x_i

Our API will follow a combination of both logics. For the ODE solver part we will define functors that have APIs similar to mc-stan and for the samplers we will follow TensorFlow's API.

6.3 Milestones

The project can be divided into discrete milestones that should be followed throughout the project. Parts indicated with asterisk (*) will be optional for completion. Moreover we divide 100 points of time on each individual milestone. Below we give a detailed table of contents for each individual milestone. All milestones include documentation and testing.

- **Milestone I (6 weeks): ODE Solvers (C++ backend).** The resulting solvers will constitute a sub-directory `include/ode_solve`.
  - Implement `GenericODESolver`.[17] (1 week)
  - Euler Method. (0.5 week)
  - RK4 Method. (1 week)
  - Leapfrog Method. (0.5 week)
  - Collocation Methods (classical* and due to [LSV18]) (1.5 week)
  - Implementation of boundary oracles (for convex polytopes). (1.5 week)
    - Implementation of generic numerical equation solvers (Newton-Raphson)
    - Generalization of Cyrus-Beck algorithm on the existing GeomScale API.
- **Milestone II (4 weeks): Samplers (C++ backend).** The resulting samplers will be added to `samplers/samplers.h`
  - Extension of BW Sampler for general distributions. (0.5 week)
  - Extension of HR Sampler for general distributions. (0.5 week)
  - Implementation of HMC Samplers (invoking the various solvers in form of functors). (1 week)[18]
    - Implementation of Classical HMC
    - Implementation of HMC-ECS due to [DQK+19].*

[17] Here discussion is needed with the mentors since large part of the project will be based on this design.

[18] The period of 1 week refers to the Classical HMC. If we are ahead of timeline, we can devote another 1-1.5 week to implement the optional samplers.

    - Implementation of the RMM sampler based on ULD due to [SL19]*.
  - Miscellaneous Applications.*
- **Milestone III (2 weeks): (R frontend).** Port functionality to R using `Rcpp`.
  - Port ODE Solvers. (0.75 week)
  - Port Samplers. (0.75 week)
  - High-level (business logic) documentation (0.5 week) like `mc-stan` and Tensorflow documentation resources.
  - Extra Testing.*
- **Milestone IV (Bonus beyond GSoC).** Miscellanious
  - Comparison against `mc-stan` for non-truncated sampling from log-concave densities. For the time being `mc-stan` does not support sampling from truncated densities so addition of boundary oracles there would be a plus.
  - Implementing a solver for real root finding, e.g. [Pan15].*
  - Python Bindings.*
  - Further Research Collaboration (open).*

7 Timeline

A buffer period of one week is left to tackle unpredictable personal delays. The milestones delineated above can be framed in terms of time in a timeline. Apart from the basic milestones, community bonding, familiarization with the GeomScale project as well as reading multiple research articles are needed to proceed to the actual project. Below we give an indicative timeline for the proposed project. There is a total of 12 weeks after the start of GSoC. The breakdown of the individual stages into Milestones on a work-week level is presented in Section 6.3.

- Stage 0: (Acceptance — Start of GSoC)
  - Familiarization with the codebase.
  - Familiarization with Rcpp.
  - Community bonding with mentors.
  - Study of research articles [SL19, LSV18] for log-concave distribution sampling.
  - Implementation of a baseline sampler (e.g. extend BW) and a baseline solver (e.g. Euler).
- Stage 1: (6 weeks)
  - Implementation of Milestone I.
  - Documentation and testing.
- Stage 2: (4 Weeks)
  - Implementation of Milestone II.
  - Documentation and testing.
- Stage 3: (2 Weeks)
  - Implementation of Milestone III.
  - Documentation and testing.

The above is subject to Personal Commitments delineated in Section 11.

8 Development Workflow

8.1 Workflow

The integration of the proposed features to GeomScale will be done via the Pull-Request Workflow. More specifically, at each milestone I prospect to submit a pull request from my fork to the main GeomScale project. My work will be done in a separate branch, specific to my project. My communication with the mentors will be done through GitHub issues, e-mail, Skype and gitter. A plausible communication plan can be SCRUM-motivated: calls between intervals of 1–2 weeks with e-mail/chat communication in the meantime. Keeping a backlog through GitHub projects (a la Kanban) or through Google Docs can be very helpful and coordinating for both me and the mentor(s).

8.2 Mentors

I have been in touch with Vissarion Fisikopoulos through e-mail and gitter after solving the required challenges. I am open to collaboration with all the mentors referred to the wiki page.

9 Development Tools

9.1 Deployment

The programming languages that GeomScale is developed are C++, R and Python. We will use the predefined toolchain for building packages and deploying, that is GNU Make and `g++` for C++, `devtools` for R and `cython` for Python.

9.2 Documentation

The project will be documented using the following packages, which are already used in GeomScale

- Comments for the backbone code in C++.
- `roxygen` for R.
- Docstrings for Python.
- Markdown for the business-logic-related artifacts.

Moreover, external documentation in forms of README/tutorials will be added accordingly if necessary.

9.3 Testing

Testing is essential for every software project. The test-driven-development approach (TDD) has been successfully followed in various software engineering projects. In the case of GeomScale, we will use the following frameworks to develop tests that are analogous to the existing ones.

- `doctest`[19] for C++. The `doctest` library is a standard C++ library for TDD.
- `testhat`[20] for the R language.
- `pytest`[21] for the Python langugage.

whereas the contents of the tests will contain

- Testing the samplers individually, i.e. by approximating polytope volumes compared to the existing samplers.
- Benchmarking the various algorithms used for the development.
- Testing the ODE solvers.

The existing tests already cover the basic functionality of the project and the new tests can be based on the existing ones.

10 Indicative Applications of the Project

10.1 Solvers for ODEs of the form $\dot{x}(t) = F(x, t)$

The primary goal of this project is to develop the ODE solvers required to solve the differential equation $\dot{x} = F(x, t)$. The provided solvers can be used — of course — for other applications in general numerical ODE solving.

10.2 Faster Inference for Latent Variable Models

Many statistical models involve both observed and latent variables — denoted by X and Z respectively — for which the parameters need to be inferred. A classical way to do parameter inference in such models is through the Expectation-Maximization (EM) algorithm, since the direct optimization of the maximum likelihood problem is intractable [SSBD14, Bis06]. The EM algorithm approximates the likelihood function with the auxiliary function

$$Q(\theta|\theta_0) = \mathbb{E}_{p(Z|X,\theta_0)}\left[\log p(X, Z|\theta)\right] = \sum_Z p(Z|X, \theta_0) \log p(X, Z|\theta) \tag{79}$$

where θ is the set of current parameters and θ_0 is the set of parameters from the previous iteration and relies on the iterative maximization of that function, which serves as a lower bound to the exact log-likelihood which is

$$\log p(X|\theta) = \log\left(\sum_Z p(X, Z|\theta)\right)$$

Latent variable models have been extensively studied in statistical physics, network science and statistics in general. An interesting line of work[22] can be found at [KL12, KL11, Kad09, Sta71].

It is clear enough that Eq. 79 cannot be inferred exactly for multiple reasons, the first one being the absense of a convenient form for optimization. For that reason, many authors

[19] https://github.com/onqtam/doctest
[20] https://github.com/r-lib/testthat
[21] https://docs.pytest.org/en/latest/
[22] Related to my thesis work

prefer to do the summation using Monte Carlo Algorithms — such as the classical Metropolis-Hastings, HMC, BW, HR and so on — approximate the auxiliary function with the ergodic sum and then use Stochastic Gradient Descent (or any other applicable optimizer) to find the new vector of parameters.

Implementing a faster ODE solver — due to [LSV18] — and the respective sampler of [SL19], we can solve statistical inference problems very fast. In multiple cases, the posterior distributions that are used are log-concave and therefore efficient sampling and computation are crucial to many modern applications.

Example Application. This application was one of the initial ideas I had for my Thesis. We are given a bipartite graph with sets C (celebrities) and U (rest of the users)[23]. Each user u is modelled by an unknown (latent) d-dimensional vector F_u of hobbies (or general interests) where a certain coordinate F_{iu} of a user is 1 if the user engages in the interest and 0 otherwise. Each interest F_{ui} appears independently (per coordinate) with proability p_i. Each celebrity has a public profile where he/she demonstrates a d-dimensional feature vector f_u. Moreover, suppose that we know the edges of the graph (think of a scenario where users like pages on Facebook). Now suppose that a user u follows a celebrity c with probability equal to

$$p(u, c) = \frac{1}{1 + \exp(-F_u^T W f_c)} \tag{80}$$

where W is an association matrix between the interests of the users. The likelihood maximization problem for finding the parameters p_i as well as the matrix W is intractable since one has to sum over exponential number of events in order to eliminate the latent features. For this reason we reside in optimizing the variational lower bound

$$\mathbb{E}_{Q(F_U)}[\log p(G, F_U, F_C)] = \sum_{F_U} Q(F_U) \log(G, F_U, F_C) \tag{81}$$

where Q is a general (variational) distribution on the latent features. Work in statistical physics [Kad09, Sta71] has addressed such problems by imposing a variational distribution Q that factorizes over the latent variables. More specifically

$$Q = \prod_{i=1}^{d} \prod_{u \in U} \phi_{iu}^{F_{iu}} (1 - \phi_{iu})^{1 - F_{iu}} \tag{82}$$

Now, the computation of the expected value can be done through MCMC methods. More precisely, we pick a user u uniformly at random, draw a new feature vector with parameters ϕ_{iu} and transist to the next candidate state of the feature matrix F'_U (noted by $F_U(t)$ with probability $\min\{1, Q(F'_U)/Q(F_U)\}$. The variational bound is now calculated as the ergodic mean over the samples $F_u(t)$, that is

$$\frac{1}{T} \sum_{t=1}^{T} \sum_{u,c} \left(A(u, c) \log \sigma(F_u(t)^T W f_v) + (1 - A(u, c)) \log(1 - \sigma(F_u(t)^T W f_v)) \right) \tag{83}$$

[23] Assume that we know the celebrities of the network

where $\sigma(x) = 1/(1+\exp(-x))$. Then the variational parameters ϕ_{iu} can be updated via using proj-SGD (for example) on the computed ergodic mean. After that, the feature distribution log-likelihood under the variational expectation becomes

$$\sum_{u\in U}\sum_{i=1}^{d}\left(\phi_{iu}\log(1-p_i)+(1-\phi_{iu})\log(1-p_i)\right) \tag{84}$$

Hence the new values for the feature distribution are update as $p_i = \frac{1}{|U|}\sum_{u\in U}\phi_{iu}$. The procedure repeats until good fits are obtained. Such problems, where the variational expectation of the likelihood (also called Evidence Lower Bound or ELBO) cannot be calculated can arise often in online social networks. Having chains for which the ergodic mean is calculated without large complexity is crucial to many models. The same can apply to a more general predictor, such as a neural net.

11 Personal Commitments

I am delivering my thesis defense in June 2020 in order to obtain the Electrical and Computer Engineering Diploma from the National Technical University of Athens. I will submit a paper to NIPS 2020 (the paper is almost finished from a technical and theoretical viewpoint and writing needs to be polished). Moreover, I have completed 1/3 of my dissertation text (thesis), which I prospect to finish the following months (upon the completion of the draft). I also have a compilers project which I prospect to finish before the official start of GSoC 2020. Moreover, I am relocating to the United States from August in order to pursue my Ph.D. degree at Cornell under a fully-funded fellowship. Relocation procedures are not yet known to me (and may be delayed due to the COVID-19 outbreak), so there is a slight chance that they may take out manageable time. For this reason I have added buffer weeks such in order to tackle unexpected events.

Finally, I have been able to survive Google Summer of Code 2018 with a semester of 8 classes and 48 assignments in total, with a very difficult project (no prior work existing), the most difficult combination of courses available to my School (Software, Signal Processing, Systems, Math) and a start-up project. I successfully completed both the project (which was funded for the next year) and aced my exams. Upon the end of GSoC, my project had circa 540 commits which amounts for 9 commits per workday. Information can be requested from my referees as well.

Referees (Alphabetical Order). I below give a list of referees in alphabetical order:

- Dimitris Fotakis (Assoc. Prof. at NTUA, Thesis advisor) — `fotakis@cs.ntua.gr`.
- Diomidis Spinellis (Prof. at AUEB, ex-GSoC mentor, Research advisor at BALab) — `dds@aueb.gr`.

Eligibility. I am eligible for applying to the program since I have participated only once as a student and have valid studentship (undegraduate) status by 27 April, 2020.

References

[AP98] Uri M Ascher and Linda R Petzold. *Computer methods for ordinary differential equations and differential-algebraic equations*, volume 61. Siam, 1998.

[BB88] Jonathan Barzilai and Jonathan M Borwein. Two-point step size gradient methods. *IMA journal of numerical analysis*, 8(1):141–148, 1988.

[Bis06] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[CB78] Mike Cyrus and Jay Beck. Generalized two-and three-dimensional clipping. *Computers & Graphics*, 3(1):23–28, 1978.

[DKPR87] Simon Duane, Anthony D Kennedy, Brian J Pendleton, and Duncan Roweth. Hybrid monte carlo. *Physics letters B*, 195(2):216–222, 1987.

[DQK+19] Khue-Dung Dang, Matias Quiroz, Robert Kohn, Minh-Ngoc Tran, and Mattias Villani. *Hamiltonian Monte Carlo with energy conserving subsampling*. MIT Press, 2019.

[Kad09] Leo P Kadanoff. More is the same; phase transitions and mean field theories. *Journal of Statistical Physics*, 137(5-6):777, 2009.

[KL11] Myunghwan Kim and Jure Leskovec. Modeling social networks with node attributes using the multiplicative attribute graph model. *arXiv preprint arXiv:1106.5053*, 2011.

[KL12] Myunghwan Kim and Jure Leskovec. Multiplicative attribute graph model of real-world networks. *Internet mathematics*, 8(1-2):113–160, 2012.

[Lou15] Michael Loulakis. Stochastic processes. *Kallipos*, 2015.

[LST20] Yin Tat Lee, Ruoqi Shen, and Kevin Tian. Logsmooth gradient concentration and tighter runtimes for metropolized hamiltonian monte carlo. *arXiv preprint arXiv:2002.04121*, 2020.

[LSV18] Yin Tat Lee, Zhao Song, and Santosh S Vempala. Algorithmic theory of odes and sampling from well-conditioned log-concave densities. *arXiv preprint arXiv:1812.06243*, 2018.

[Pan15] Victor Y Pan. Simple and nearly optimal polynomial root-finding by means of root radii approximation. In *Special Sessions in Applications of Computer Algebra*, pages 329–340. Springer, 2015.

[SL19] Ruoqi Shen and Yin Tat Lee. The randomized midpoint method for log-concave sampling. In *Advances in Neural Information Processing Systems*, pages 2098–2109, 2019.

[Smi96] Robert L Smith. The hit-and-run sampler: a globally reaching markov chain sampler for generating arbitrary multivariate distributions. In *Proceedings Winter Simulation Conference*, pages 260–264. IEEE, 1996.

[SSBD14] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[Sta71] HE Stanley. Mean field theory of magnetic phase transitions. *Introduction to Phase Transitions and Critical Phenomena*, 1971.

A Source Code of Tasks

A.1 Easy

```
#!/usr/bin/env r
# Easy challenge


library(ggplot2)
library(volesti)

d <- 10
P <- volesti::gen_cube(d, 'H')
points <- volesti::sample_points(P=P, n=1000)
g <- plot(ggplot(data.frame(x=points[1,], y=points[2,])) +
  geom_point(aes(x=x, y=y), color="blue") +
  coord_fixed(xlim=c(-1,1), ylim=c(-1,1)) +
  ggtitle(sprintf("Example␣Output␣of␣Easy␣Task")))
```

A.2 Medium

```
/* Medium-Difficulty task for GeomScales GSOC2020
 *
 * In the current task the ball sampling algorithm
 * is implemented so as to sample from a log-concave
 * distribution g(x) = exp(-f(x)) where f(x) is a
 * strongly convex function.
 *
 * Author: Marios Papachristou (papachristoumarios@gmail.com)
 *
 */

#include <iostream>
#include <cmath>
```

```
#include <boost/numeric/ublas/vector.hpp>
#include <boost/numeric/ublas/io.hpp>
#include <random>
using namespace boost::numeric::ublas;

typedef vector<double> vd;

vd get_random_vector(unsigned int number_of_dimensions, std::
   uniform_real_distribution<double> &dist, std::
   default_random_engine &gen, double scale) {
  vd x = vd(number_of_dimensions);

  for (int i = 0; i < number_of_dimensions; i++) {
    x(i) = scale * dist(gen);
  }

  return x;
}

vd get_random_normal_vector(unsigned int number_of_dimensions,
   std::normal_distribution<double> &dist, std::
   default_random_engine &gen, double scale) {
  vd x = vd(number_of_dimensions);

  for (int i = 0; i < number_of_dimensions; i++) {
    x(i) = scale * dist(gen);
  }

  return x;
}


bool in_poly(vd &x) {
  // Polytope oracle. Returns true if x is in K
  double tmp = norm_2(x);
  return tmp * tmp <= 1;
}

double f(vd &x) {
  // Strongly convex function oracle for f(x) = 2 ||x||^2
  double tmp = norm_2(x);
  return 0.5 * tmp * tmp;
}

double g(vd &x) {
```

```
  // Log concave distribution modulo normalization constant 1/Z
  double res = f(x);
  return exp(-res);
}

double min(double x, double y) {
  return x <= y ? x : y;
}

vd uniform_sphere_mulller(double &delta, unsigned int &
   number_of_dimensions, std::uniform_real_distribution<double>
    &udist, std::normal_distribution<double> &ndist, std::
   default_random_engine &gen) {
  std::default_random_engine ngen;

  vd y = get_random_normal_vector(number_of_dimensions, ndist,
     gen, 1);
  double y_norm = norm_2(y);

  y = y / y_norm;

  double rho = delta / number_of_dimensions * get_random_vector
     (1, udist, gen, 1)(0);

  return rho * y / y_norm;

}

vd next(vd &x, double &delta, unsigned int &
   number_of_dimensions, std::uniform_real_distribution<double>
    &dist, std::normal_distribution<double> &ndist, std::
   default_random_engine &gen) {
  // Get next sample given x
  vd y, r;

  do {
    r  = uniform_sphere_mulller(delta, number_of_dimensions,
       dist, ndist, gen);

    y = x + r;
  } while (!in_poly(y));

  double prob = min(1, g(y) / g(x));

  std::bernoulli_distribution bern(prob);
```

```
  return bern(gen) ? y : x;
}


int main() {
  // Number of dimensions of the distribution parametrized by f
     (x)
  unsigned int number_of_dimensions = 2;

  // Number of samples to be drawn using ball-sampling
  unsigned int number_of_samples = 10000;

  // Precision (distance between consecutive samples)
  double delta = 0.01;

  // Initialize random generator and distribution for sampling
     in U[0, 1]
  std::default_random_engine generator;
  std::uniform_real_distribution<double> distribution(-1.0,1.0)
     ;
  std::normal_distribution<double> ndist(0.0, 1.0);

  std::cout << "Ball-walk sampling procedure" << std::endl;
  vd x;
  x = get_random_vector(number_of_dimensions, distribution,
     generator, 1);
  for (int i = 0; i < number_of_samples; i++) {
      std::cout << "Sample #" << i << " is " << x << std::endl;
      x = next(x, delta, number_of_dimensions, distribution,
         ndist, generator);
  }

}
```

A.3 Hard

```
/* Hard-Difficulty task for GeomScales GSOC2020
 *
 * Here we optimize a strongly convex function f(x)
 * using the BB algorithm which approximates the inv. Hessian
 * of f times the grad of f via solving a least squares problem
 * Example on the function f(x) = x^T x
 * Author: Marios Papachristou (papachristoumarios@gmail.com)
 *
```

```
 */

#include <iostream>
#include <cmath>
#include <boost/numeric/ublas/vector.hpp>
#include <boost/numeric/ublas/io.hpp>
#include <random>
using namespace boost::numeric::ublas;

typedef vector<double> vd;

vd get_random_vector(unsigned int number_of_dimensions, std::
   uniform_real_distribution<double> &dist, std::
   default_random_engine &gen, double scale) {
  vd x = vd(number_of_dimensions);

  for (int i = 0; i < number_of_dimensions; i++) {
    x(i) = scale * dist(gen);
  }

  return x;
}

double f(vd x) {
  double temp = norm_2(x);
  return temp * temp;
}

vd grad_f(vd x) {
  return 2 * x;
}

int main() {

  // Number of dimensions of the distribution parametrized by f
     (x)
  unsigned int number_of_dimensions = 2;

  // Tolerance factor for convergence checks
  double eps = 1e-6;

  // Bounds to sample initial point for GD
  double xlow = -100.0;
  double xhigh = 100.0;
```

```
// Initialize random generator and distribution for sampling
   in U[0, 1]
std::default_random_engine generator;
std::uniform_real_distribution<double> distribution(xlow,
   xhigh);
std::bernoulli_distribution bern(0.5);

std::cout << "Gradient descent with BB step procedure" << std
   ::endl;
int counter = 1;

// Variable initialization
vd x = get_random_vector(number_of_dimensions, distribution,
   generator, 1);
vd x_prev = vd(x);
vd derivative = get_random_vector(number_of_dimensions,
   distribution, generator, 1);
vd derivative_prev = vd(x);
double ak_1, ak_2;
vd s = vd(x);
vd y = vd(x);
double eta = 1e-6;

// BB Gradient Descent
do {
  derivative = grad_f(x);
  std::cout << "Iteration #" << counter << " :";
  std::cout << "Value of f(x) = " << f(x) << " ";
  std::cout << "Grad of f = " << derivative << std::endl;
  s = x - x_prev;
  y = derivative - derivative_prev;

  // Calculate helper learning rates
  ak_1 = inner_prod(s, s) / inner_prod(y, y);
  ak_2 = inner_prod(s, y) / inner_prod(y, y);

  if (counter > 1) {
    std::cout << "Candidate Learning Rates: " << ak_1 << ", "
        << ak_2 << std::endl;
    // Pick a learning rate with equal probability
    eta = bern(generator) ? ak_1 : ak_2;
  }

  x_prev = x;
  x = x - eta * derivative;
```

```
    derivative_prev = derivative;
    counter++;

  } while(norm_2(x - x_prev) > eps);
}
```