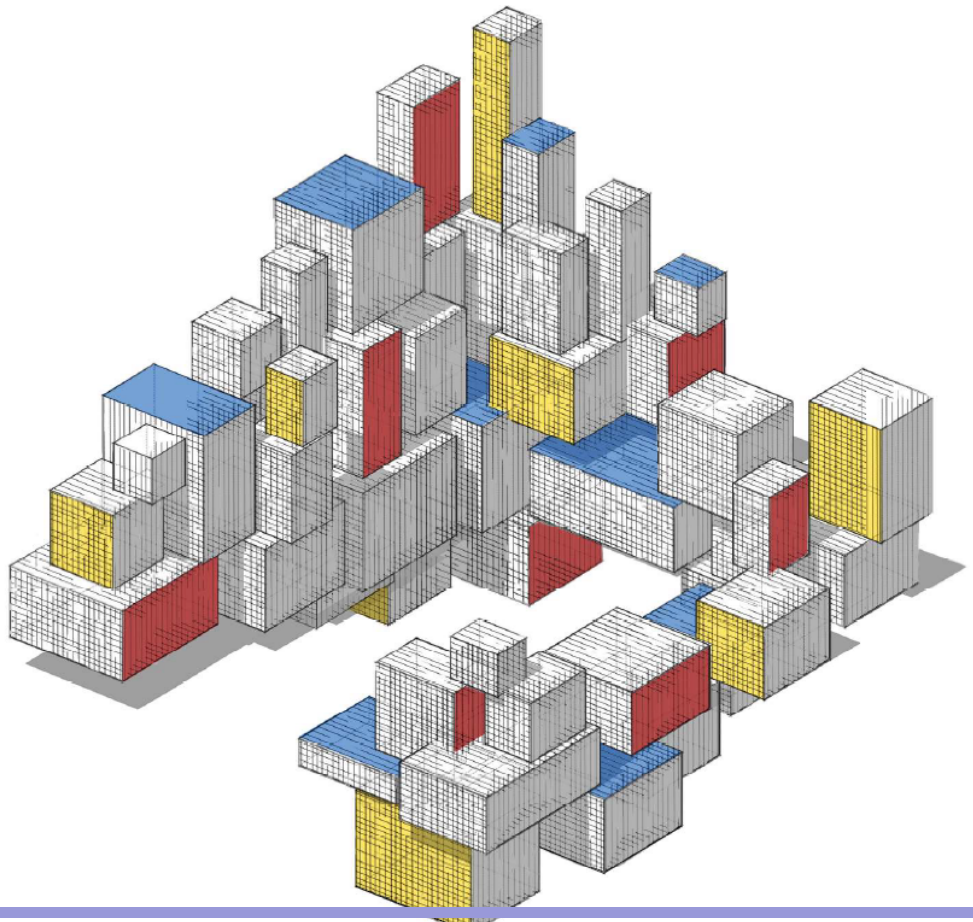
Monte Carlo for Optimization



Purpose

In this lecture we discuss two Monte Carlo methods for optimization, including:

- Simulated Annealing
- Stochastic Optimization

Such randomized algorithms can be useful for solving optimization problems that involve

- many local optima
- complicated constraints
- a mix of continuous and discrete variables
- *noisy* objective functions (e.g., that are obtained via Monte Carlo simulation)

Simulated Annealing

Simulated annealing is a Monte Carlo technique for minimization that emulates the physical state of atoms in a metal when the metal is heated up and then slowly cooled down. When the cooling is performed very slowly, the atoms settle down to a minimum-energy state.

Denoting the state as $\boldsymbol{x}$ and the energy of a state as $S(\boldsymbol{x})$, the probability distribution of the (random) states is described by the Boltzmann pdf

$$f(\boldsymbol{x}) \propto \mathrm{e}^{-\frac{S(\boldsymbol{x})}{kT}}, \quad \boldsymbol{x} \in \mathcal{X},$$

where k is Boltzmann's constant and T is the temperature.

Gibbs Distribution

Suppose that $S(\boldsymbol{x})$ is an arbitrary function to be minimized, with $\boldsymbol{x}$ taking values in some discrete or continuous set $\mathcal{X}$. The Gibbs pdf corresponding to $S(\boldsymbol{x})$ is defined as

$$f_T(\boldsymbol{x}) = \frac{\mathrm{e}^{-\frac{S(\boldsymbol{x})}{T}}}{z_T}, \quad \boldsymbol{x} \in \mathcal{X},$$

provided that $z_T := \sum_{\boldsymbol{x}} \exp(-S(\boldsymbol{x})/T) < \infty$.

As $T \to 0$, the pdf becomes more and more peaked around the set of global minimizers of S.

The idea of simulated annealing is to create a sequence of points $\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots$ that are approximately distributed according to pdfs $f_{T_1}(\boldsymbol{x}), f_{T_2}(\boldsymbol{x}), \ldots$, where $T_1, T_2, \ldots$ is a sequence of "temperatures" that decreases (is "cooled") to 0 — known as the annealing schedule.

Simulated Annealing

If each $\boldsymbol{X}_t$ were sampled *exactly* from f_{T_t}, then $\boldsymbol{X}_t$ would converge to a global minimum of $S(\boldsymbol{x})$ as $T_t \to 0$. However, in practice sampling is *approximate* and convergence to a global minimum is not assured.

Algorithm 1: Simulated Annealing

---

input: Annealing schedule $T_0, T_1, \ldots,$, function S, initial value $\boldsymbol{x}_0$.

output: Approximations to the global minimizer $\boldsymbol{x}^*$ and minimum value $S(\boldsymbol{x}^*)$.

1 Set $\boldsymbol{X}_0 \leftarrow \boldsymbol{x}_0$ and $t \leftarrow 1$.

2 **while** not stopping **do**

3 Approximately simulate $\boldsymbol{X}_t$ from $f_{T_t}(\boldsymbol{x})$.

4 $t \leftarrow t + 1$

5 **return** $\boldsymbol{X}_t, S(\boldsymbol{X}_t)$

Approximate Sampling from the Gibbs Distribution

A popular annealing schedule is geometric cooling, where $T_t = \beta T_{t-1}$, $t = 1, 2, \ldots$, for a given initial temperature T_0 and a cooling factor $\beta \in (0, 1)$. Appropriate values for T_0 and β are problem-dependent.

A possible stopping criterion is to stop after a fixed number of iterations, or when the temperature is "small enough".

Approximate sampling from a Gibbs distribution is most often carried out via Markov chain Monte Carlo.

For each iteration t, the Markov chain should theoretically run for a large number of steps to accurately sample from the Gibbs pdf f_{T_t}.

However, in practice, one often only runs a *single* step of the Markov chain, before updating the temperature, as in Algorithm 2 below.

Algorithm 2: Simulated Annealing with a Random Walk Sampler

input: Objective function S, starting state $\boldsymbol{X}_0$, initial temperature T_0, number of iterations N, symmetric proposal density $q(\boldsymbol{y} \mid \boldsymbol{x})$, constant β.

output: Approximate minimizer and minimum value of S.

for $t = 0$ **to** $N - 1$ **do**
  Simulate a new state $\boldsymbol{Y}$ from the symmetric proposal $q(\boldsymbol{y} \mid \boldsymbol{X}_t)$.
  if $S(\boldsymbol{Y}) < S(\boldsymbol{X}_t)$ **then**
    $\boldsymbol{X}_{t+1} \leftarrow \boldsymbol{Y}$
  else
    Draw $U \sim \mathcal{U}(0, 1)$.
    if $U \leqslant \mathrm{e}^{-(S(\boldsymbol{Y}) - S(\boldsymbol{X}_t))/T_t}$ **then**
      $\boldsymbol{X}_{t+1} \leftarrow \boldsymbol{Y}$
    else
      $\boldsymbol{X}_{t+1} \leftarrow \boldsymbol{X}_t$
  $T_{t+1} \leftarrow \beta T_t$
return $\boldsymbol{X}_N$ and $S(\boldsymbol{X}_N)$

Metropolis–Hastings Simulated Annealing

To sample from a Gibbs distribution f_T, the algorithm above uses a random walk Metropolis–Hastings sampler.

The acceptance probability of a proposal $\boldsymbol{y}$ is in this case:

$$\alpha(\boldsymbol{x}, \boldsymbol{y}) = \min\left\{\frac{\mathrm{e}^{-\frac{1}{T}S(\boldsymbol{y})}}{\mathrm{e}^{-\frac{1}{T}S(\boldsymbol{x})}},\ 1\right\} = \min\left\{\mathrm{e}^{-\frac{1}{T}(S(\boldsymbol{y})-S(\boldsymbol{x}))},\ 1\right\}.$$

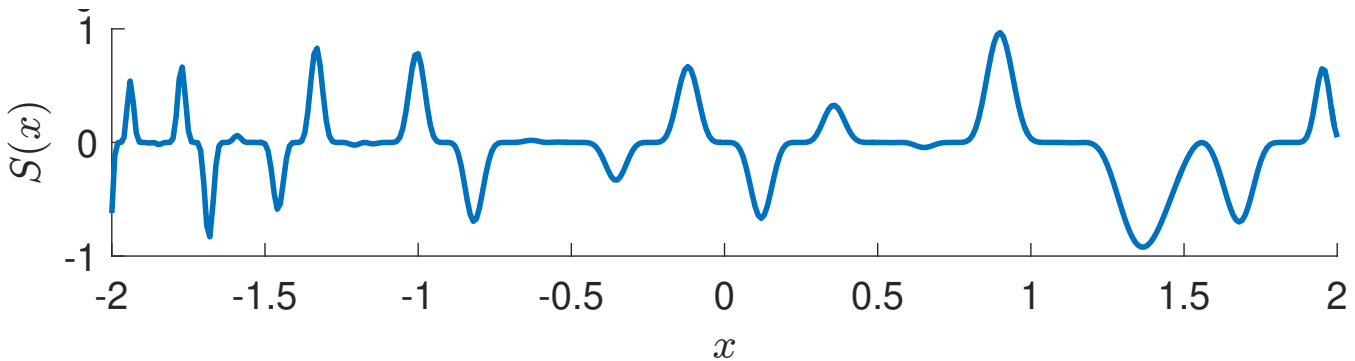
Hence, if $S(\boldsymbol{y}) < S(\boldsymbol{x})$, then the proposal is aways accepted.

Otherwise, the proposal is accepted with probability $\exp(-\frac{1}{T}(S(\boldsymbol{y}) - S(\boldsymbol{x})))$.

Simulated Annealing for Minimization

We wish to minimize the following "wiggly" function:

$$S(x) = \begin{cases} -\mathrm{e}^{-x^2/100} \sin(13x - x^4)^5 \sin(1 - 3x^2)^2, & \text{if } -2 \leqslant x \leqslant 2, \\ \infty, & \text{otherwise.} \end{cases}$$



The function has many local minima and maxima, with a global minimum around 1.4.

Simulated Annealing

As the temperature decreases, the Gibbs pdf converges to the pdf that has all its mass concentrated at the minimizer of S.

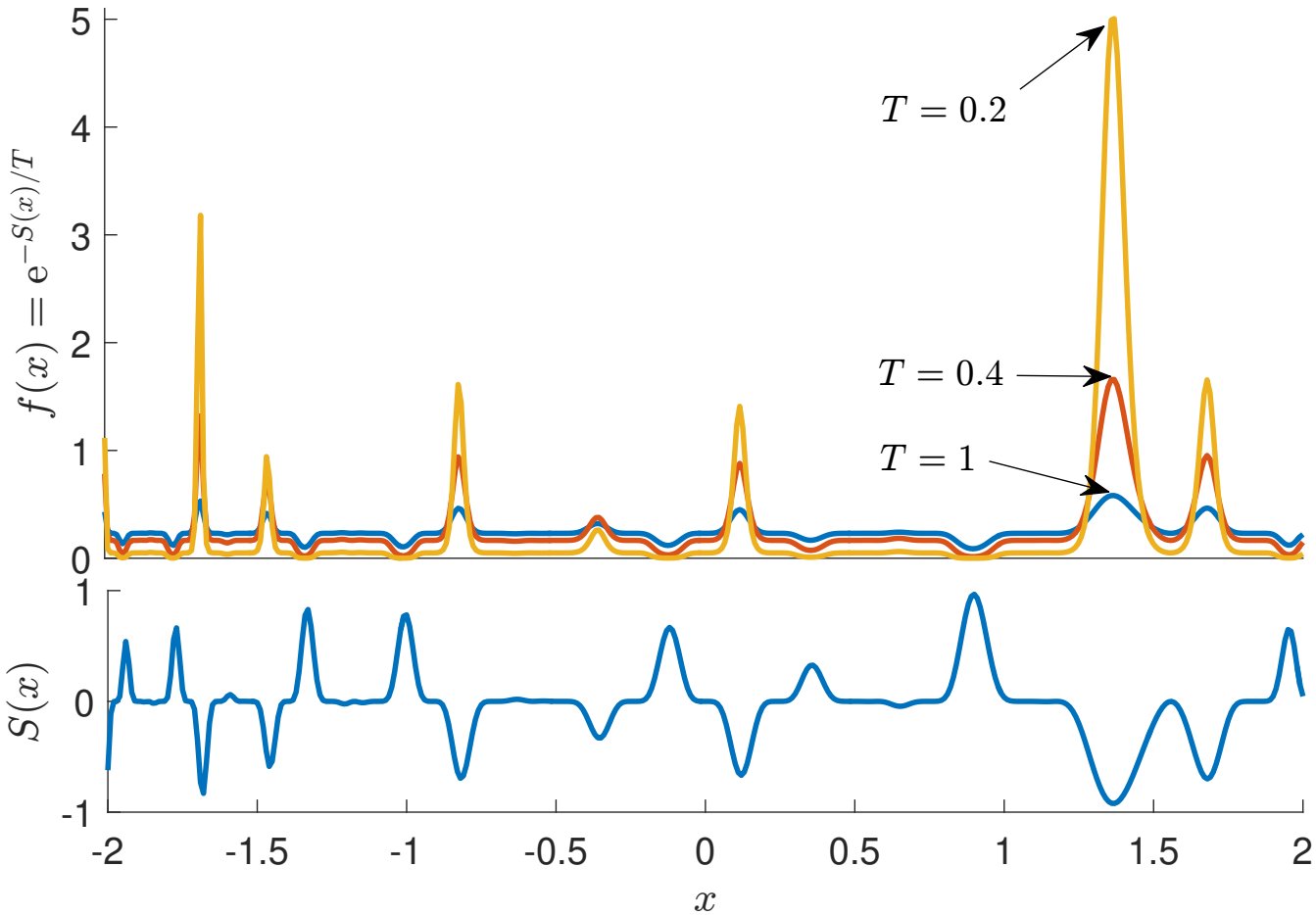


Figure: Upper panel: Gibbs pdfs for temperatures $T = 1, 0.4.0.2$.

Simulated Annealing

Possible stopping criteria:

- Stopping after a fixed number of iterations
- Stop when the temperature is lower than some threshold, e.g. 10^{-3}.
- Stop when consecutive function values are closer than some distance ε to each other
- Stop when the best found function value has not changed over a fixed number d of iterations.

In the following Python code, we use the second stopping criterion.

For a "current" state x, the proposal state Y is here drawn from the $\mathcal{N}(x, 0.5^2)$ distribution.

We use geometric cooling with decay parameter $\beta = 0.999$ and initial temperature $T_0 = 1$.

We set the initial state to $x_0 = 0$.

simann.py

```
import numpy as np
import matplotlib.pyplot as plt

def wiggly(x):
    y = -np.exp(x**2/100)*np.sin(13*x-x**4)**5*np.sin(1-3*x**2)**2
    ind = np.vstack((np.argwhere(x<-2),np.argwhere(x>2)))
    y[ind]=float('inf')
    return y

S = wiggly,  beta = 0.999,  sig = 0.5,  T=1
x= np.array([0]), xx=[]
Sx=S(x)
while T>10**(-3):
    T=beta*T
    y = x+sig*np.random.randn()
    Sy = S(y)
    alpha = np.amin((np.exp(-(Sy-Sx)/T),1))
    if np.random.uniform()<alpha:
        x=y
        Sx=Sy
    xx=np.hstack((xx,x))

print('minimizer = {:3.3f}, minimum ={:3.3f}'.format(x[0],Sx[0]))
plt.plot(xx)
```

```
minimizer = 1.365, minimum = -0.958
```

After initially fluctuating wildly, the sequence of states settles down to a value around 1.37, with $S(1.37) = -0.92$, corresponding to the global optimizer and minimum, respectively.

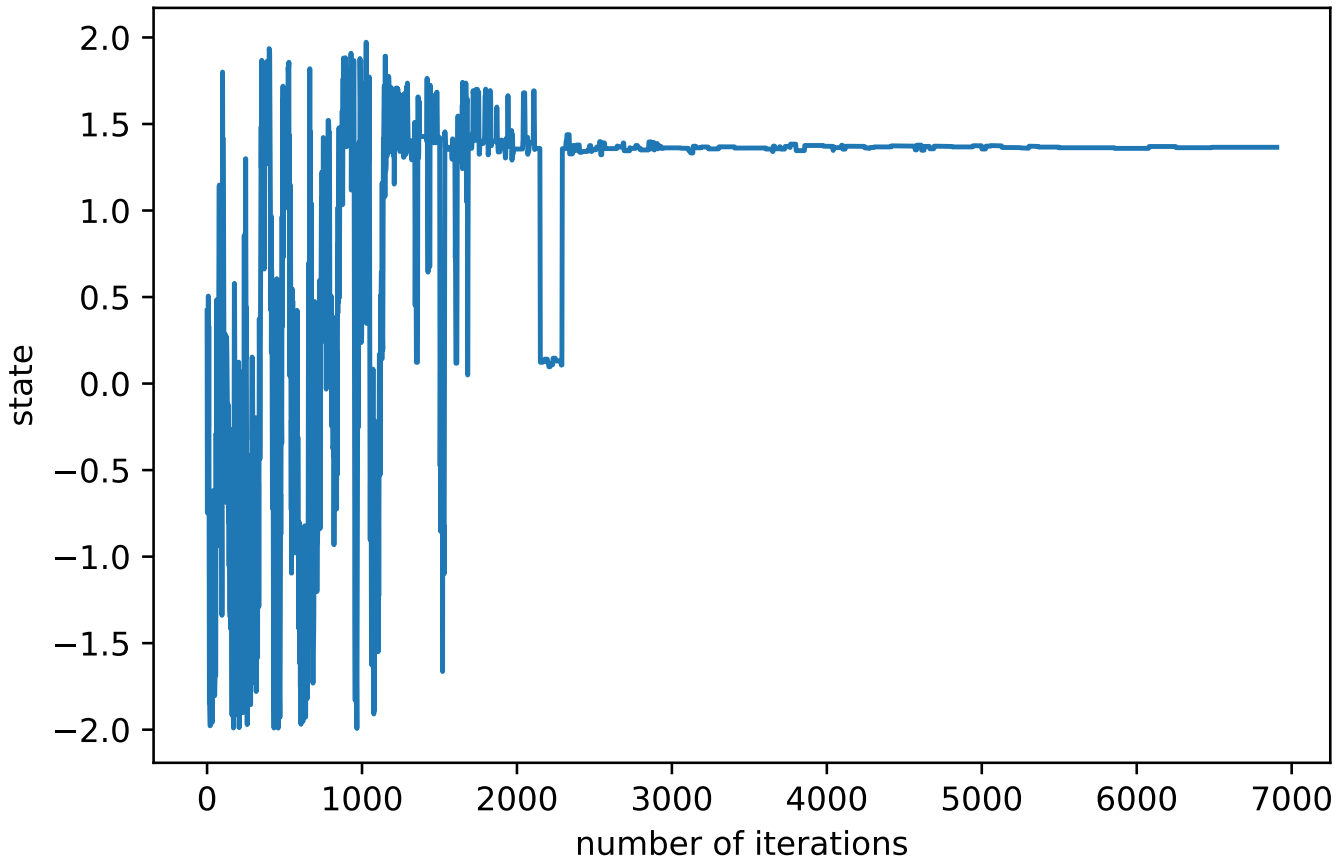


Figure: Typical states generated by the simulated annealing algorithm.

Noisy Optimization

In noisy optimization, the objective function is unknown, but estimates of function values are available, e.g., via simulation.

For example, to find an optimal prediction function g in supervised learning, the exact risk $\ell(g) = \mathbb{E}\,\mathrm{Loss}(Y, g(\boldsymbol{x}))$ is usually unknown and only estimates of the risk are available. Optimizing the risk is thus typically a noisy optimization problem.

Noisy optimization features prominently in simulation studies where the behavior of some system (e.g., vehicles on a road network) is simulated under certain parameters (e.g., the lengths of the traffic light intervals) and the aim is to choose those parameters optimally (e.g., to maximize the traffic throughput).

Stochastic Approximation

Suppose the goal is to minimize a function S, where S is unknown, but an estimate of $S(\boldsymbol{x})$ can be obtained for any choice of $\boldsymbol{x} \in \mathcal{X}$.

Stochastic approximation mimics the classical gradient descent method by replacing a deterministic gradient with an estimate $\widehat{\nabla S}(\boldsymbol{x})$.

A simple estimator for the i-th component of $\nabla S(\boldsymbol{x})$ (that is, $\partial S(\boldsymbol{u})/\partial x_i$), is the central difference estimator

$$\frac{\widehat{S}(\boldsymbol{x} + \boldsymbol{e}_i\,\delta/2) - \widehat{S}(\boldsymbol{x} - \boldsymbol{e}_i\,\delta/2)}{\delta},$$

where $\boldsymbol{e}_i$ denotes the i-th unit vector, and $\widehat{S}(\boldsymbol{x} + \boldsymbol{e}_i\,\delta/2)$ and $\widehat{S}(\boldsymbol{x} - \boldsymbol{e}_i\,\delta/2)$ can be any estimators of $S(\boldsymbol{x} + \boldsymbol{e}_i\,\delta/2)$ and $S(\boldsymbol{x} - \boldsymbol{e}_i\,\delta/2)$, respectively.

The parameter $\delta > 0$ should be small enough to reduce the bias of the estimator, but large enough to keep its variance small.

Common Random Numbers

To reduce the variance in the central difference estimator it is important to have $\widehat{S}(\boldsymbol{x} + \boldsymbol{e}_i\,\delta/2)$ and $\widehat{S}(\boldsymbol{x} - \boldsymbol{e}_i\,\delta/2)$ positively correlated. This can for example be achieved by using common random numbers in the simulation.

In direct analogy to gradient descent methods, the stochastic approximation method produces a sequence of iterates, starting with some $\boldsymbol{x}_1 \in \mathcal{X}$, via

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \beta_t\,\widehat{\nabla S}(\boldsymbol{x}_t), \tag{1}$$

where $\beta_1, \beta_2, \ldots$ is a sequence of strictly positive step sizes. A generic stochastic approximation algorithm for minimizing a function S is thus as follows.

Algorithm 3: Stochastic Approximation

input: A mechanism to estimate any gradient $\nabla S(\boldsymbol{x})$ and step sizes $\beta_1, \beta_2, \ldots$.

output: Approximate optimizer of S.

Initialize $\boldsymbol{x}_1 \in \mathcal{X}$. Set $t \leftarrow 1$.

while a stopping criterion is not met **do**

  Obtain an estimated gradient $\widehat{\nabla S}(\boldsymbol{x}_t)$ of S at $\boldsymbol{x}_t$.

  Determine a step size β_t.

  Set $\boldsymbol{x}_{t+1} \leftarrow \boldsymbol{x}_t - \beta_t \widehat{\nabla S}(\boldsymbol{x}_t)$.

  $t \leftarrow t + 1$

return $\boldsymbol{x}_t$

Stochastic Approximation

When $\widehat{\nabla S}(\boldsymbol{x}_t)$ is an *unbiased* estimator of $\nabla S(\boldsymbol{x}_t)$ in (1), the stochastic approximation Algorithm 3 is referred to as the Robbins–Monro algorithm. When finite differences are used to estimate $\widehat{\nabla S}(\boldsymbol{x}_t)$, the resulting algorithm is known as the Kiefer–Wolfowitz algorithm.

It can be shown that, under certain regularity conditions on S, the sequence $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots$ converges to the true minimizer $\boldsymbol{x}^*$ when the step sizes decrease slowly enough to 0; in particular, when

$$\sum_{t=1}^{\infty} \beta_t = \infty \quad \text{and} \quad \sum_{t=1}^{\infty} \beta_t^2 < \infty. \tag{2}$$

In practice, one rarely uses step sizes that satisfy (2), as the convergence of the sequence will be too slow to be practical.

Stochastic Counterpart Method

An alternative approach to stochastic approximation is the stochastic counterpart method, also called sample average approximation. It can be applied in situations where the noisy objective function is of the form

$$S(\boldsymbol{x}) = \mathbb{E}\widetilde{S}(\boldsymbol{x}, \boldsymbol{\xi}), \quad \boldsymbol{x} \in \mathcal{X}, \tag{3}$$

where $\boldsymbol{\xi}$ is a random vector that can be simulated and $\widetilde{S}(\boldsymbol{x}, \boldsymbol{\xi})$ can be evaluated exactly. The idea is to replace the optimization of (3) with that of the sample average

$$\widehat{S}(\boldsymbol{x}) = \frac{1}{N} \sum_{i=1}^{N} \widetilde{S}(\boldsymbol{x}, \boldsymbol{\xi}_i), \quad \boldsymbol{x} \in \mathcal{X},$$

where $\boldsymbol{\xi}_1, \ldots, \boldsymbol{\xi}_N$ are iid copies of $\boldsymbol{\xi}$. Note that $\widehat{S}$ is a deterministic function of $\boldsymbol{x}$ and so can be optimized using any optimization algorithm. A solution to this sample average version is taken to be an estimator of a solution $\boldsymbol{x}^*$ to the original problem (3).

Example: Determining Good IS Parameters

The selection of good importance sampling parameters can be viewed as a stochastic optimization problem. We return to the example of estimating

$$\mu_b = \int_{-b}^{b} \int_{-b}^{b} \underbrace{(2b)^2 M(\boldsymbol{x})}_{H(\boldsymbol{x})} f(\boldsymbol{x})\, \mathrm{d}\boldsymbol{x} = \mathbb{E}_{f_b} H(\boldsymbol{X}),$$

where

$$M(x_1, x_2) = \mathrm{e}^{-\frac{1}{4}\sqrt{x_1^2+x_2^2}} \left(\sin\left(2\sqrt{x_1^2 + x_2^2} \right) + 1 \right), \quad (x_1, x_2) \in \mathbb{R}^2,$$

and f_b is the pdf of the uniform distribution on $[-b, b]^2$ (b is large enough, e.g., $b = 1000$). The importance sampling pdf is

$$g_\lambda(\boldsymbol{x}) = f_{R,\Theta}(r, \theta) \frac{1}{r} = \lambda \mathrm{e}^{-\lambda r} \frac{1}{2\pi} \frac{1}{r} = \frac{\lambda \mathrm{e}^{-\lambda\sqrt{x_1^2+x_2^2}}}{2\pi\sqrt{x_1^2 + x_2^2}}, \quad \boldsymbol{x} = (x_1, x_2) \in \mathbb{R}^2 \backslash \{\mathbf{0}\},$$

which depends on a free parameter λ.

In the example we chose $\lambda = 0.1$. Is this the best choice? Maybe $\lambda = 0.05$ or 0.2 would have resulted in a more accurate estimate.

The "effectiveness" of λ can be measured in terms of the variance of the importance sampling estimator, which is given by

$$\frac{1}{N}\mathbb{V}\mathrm{ar}_{g_\lambda}\left(H(\boldsymbol{X})\frac{f_b(\boldsymbol{X})}{g_\lambda(\boldsymbol{X})}\right) = \frac{1}{N}\mathbb{E}_{g_\lambda}\left[H^2(\boldsymbol{X})\frac{f_b^2(\boldsymbol{X})}{g_\lambda^2(\boldsymbol{X})}\right] - \frac{\mu^2}{N} = \frac{1}{N}\mathbb{E}_{f_b}\left[H^2(\boldsymbol{X})\frac{f_b(\boldsymbol{X})}{g_\lambda(\boldsymbol{X})}\right] - \frac{\mu^2}{N}.$$

Hence, the optimal parameter λ^* minimizes the function

$$S(\lambda) = \mathbb{E}_{f_b}\left[H^2(\boldsymbol{X}) f_b(\boldsymbol{X}) / g_\lambda(\boldsymbol{X})\right],$$

which is unknown, but can be estimated from simulation.

To solve this stochastic minimization problem, we first use stochastic approximation.

Thus, at each step of the algorithm, the gradient of $\mathbb{E}S(\lambda)$ is estimated from realizations of $\widehat{S}(\lambda) = H^2(\boldsymbol{X}) f(\boldsymbol{X}) / g_\lambda(\boldsymbol{X})$, where $\boldsymbol{X} \sim f_b$.

As in the original problem (i.e., the estimation of $\mu = \mu_\infty$), the parameter b should be large enough to avoid any bias in the estimator of λ^*, but also small enough to ensure a small variance.

The following Python code implements a particular instance of Algorithm 3. For sampling from f_b here, we used $b = 100$ instead of $b = 1000$, as this will improve the crude Monte Carlo estimation of λ^*, without noticeably affecting the bias.

The gradient of $\mathbb{E}S(\lambda)$ is estimated using the central difference estimator.

Notice how for the $S(\lambda - \delta/2)$ and $S(\lambda + \delta/2)$ the *same* random vector $\boldsymbol{X} = [X_1, X_2]^\top$ is used. This significantly reduces the variance of the gradient estimator.

The step size β_t should be such that $\beta_t \widehat{\nabla S}(\boldsymbol{x}_t) \approx \lambda_t$. Given the large gradient here, we choose $\beta_0 = 10^{-7}$ and decrease it each step by a factor of 0.99.

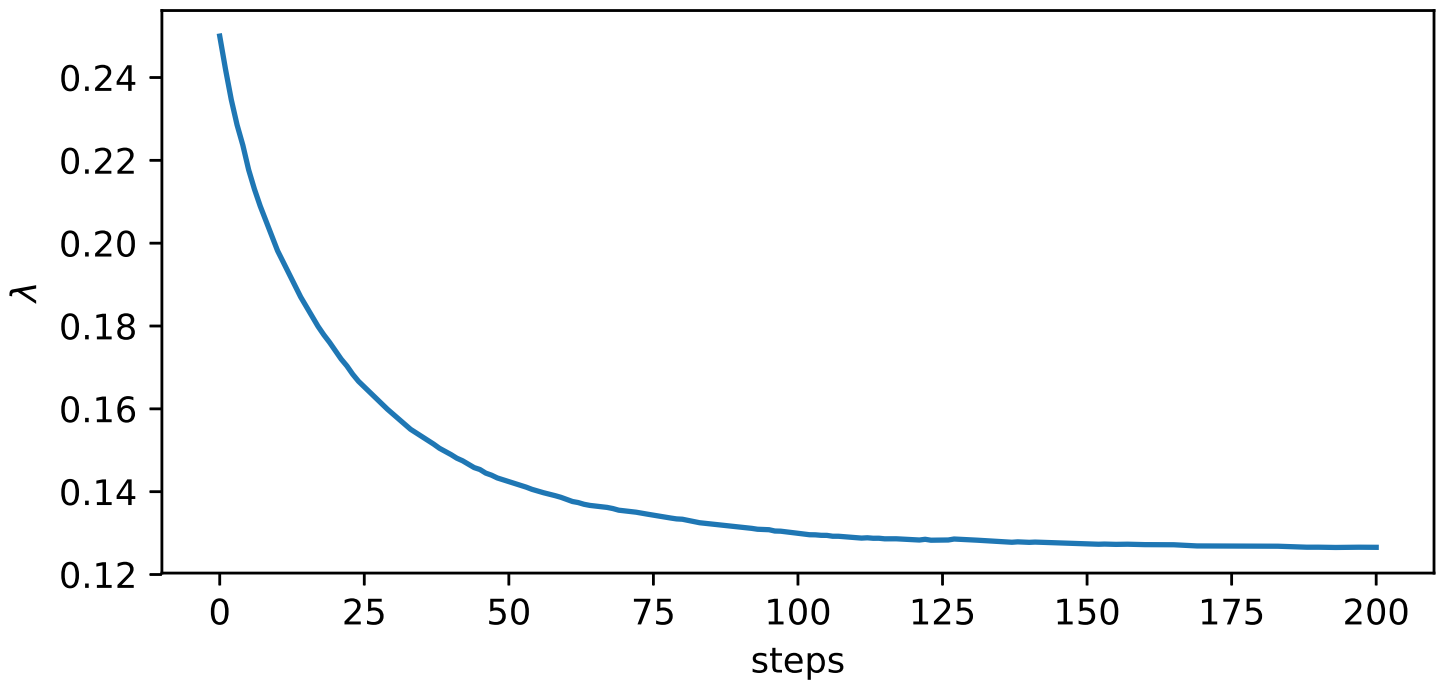
stochapprox.py

```
import numpy as np
from numpy import pi
import matplotlib.pyplot as plt

b=100    # choose b large enough, but not too large
delta = 0.01
H = lambda x1, x2: (2*b)**2*np.exp(-np.sqrt(x1**2 + x2**2)/4)*(np.sin(2*np.sqrt(x1**2+x2**2)+1)
      )*(x1**2+x2**2<b**2)
f = 1/(2*b)**2
g = lambda x1, x2, lam: lam*np.exp(-np.sqrt(x1**2+x2**2)*lam)/np.sqrt(x1**2+x2**2)/(2*pi)
beta = 10**-7   #step size very small, as the gradient is large
lam=0.25
lams = np.array([lam])
N=10**4
for i in range(200):
    x1 = -b + 2*b*np.random.rand(N,1)
    x2 = -b + 2*b*np.random.rand(N,1)
    lamL = lam - delta/2
    lamR = lam + delta/2
    estL = np.mean(H(x1,x2)**2*f/g(x1, x2, lamL))
    estR = np.mean(H(x1,x2)**2*f/g(x1, x2, lamR))  #use SAME x1,x2
    gr = (estR-estL)/delta  #gradient
    lam = lam - gr*beta  #gradient descend
    lams = np.hstack((lams, lam))
    beta = beta*0.99

lamsize=range(0, (lams.size))
plt.plot(lamsize, lams)
plt.show()
```

Stochastic Optimization

We see that the stochastic optimization algorithm produces a sequence $\lambda_t, t = 0, 1, 2, \ldots$ that tends to an approximate estimate of the optimal importance sampling parameter $\lambda^* \approx 0.125$.

This we then take as the optimal importance sampling parameter λ^*.



Stochastic Counterpart Method

We can also estimate λ^* using the stochastic counterpart approach, using the sample average

$$\widehat{S}(\lambda) = \frac{1}{N} \sum_{i=1}^{N} H^2(\boldsymbol{X}_i) \frac{f(\boldsymbol{X}_i)}{g_\lambda(\boldsymbol{X}_i)},$$

where $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N \sim_{\text{iid}} f_b$.

Once the $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N \sim_{\text{iid}} f_b$ have been simulated, $\widehat{S}(\lambda)$ is a deterministic function of λ, which can be optimized by any means.

We take the most basic approach and simply evaluate the function for $\lambda = 0.01, 0.02, \ldots, 0.3$ and select the minimizing λ on this grid.

stochcounterpart.py

```
from stochapprox import *

lams = np.linspace(0.01, 0.31, 1000)
res=[]
res = np.array(res)
for i in range(lams.size):
    lam = lams[i]
    np.random.seed(1)
    g = lambda x1, x2: lam*np.exp(-np.sqrt(x1**2+x2**2)*lam)/np.sqrt(x1
        **2+x2**2)/(2*pi)
    X=-b+2*b*np.random.rand(N,1)
    Y=-b+2*b*np.random.rand(N,1)
    Z=H(X,Y)**2*f/g(X,Y)
    estCMC = np.mean(Z)
    res = np.hstack((res, estCMC))

plt.plot(lams, res)
plt.xlabel(r'$\lambda$')
plt.ylabel(r'$\hat{S}(\lambda)$')
plt.ticklabel_format(style='sci', axis='y', scilimits=(0,0))
plt.show()
```

Stochastic Counterpart Method

The minimum value of $\widehat{S}(\lambda)$ was $1.60 \cdot 10^4$ for minimizer $\widehat{\lambda}^* = 0.12$, which is in accordance with the value obtained via stochastic approximation.

For a wide range of values (say from 0.04 to 0.15) $\widehat{S}$ stays rather flat. So any of these values could be used in an importance sampling procedure to estimate μ.