1 Global optimization

1.1 Introduction

Global optimization is the problem of locating (a good approximation to) the global minimum of a given objective function in a (given) search space that is large enough to prohibit exhaustive enumeration. When only a small sub-space of the search space can be realistically sampled within the allotted time the stochastic methods—which use some form of randomness—usually come to the fore. In the following several popular stochastic global minimization algorithms are shortly described.

1.2 Randomized local minimizers

For a differentiable optimization problem a good local minimizer would typically converge to the nearest local miminum relatively fast. Therefore a class of "quick-and-dirty" global minimizers can be constructed by simply adding some elements of random sampling to local minimizers.

1.2.1 Local minimization from several random start-points

One strategy is to start the local miminizer several times from different (quasi)random starting points within the given search space (recording the best solution). The procedure is repeated until the alloted time is exhausted.

1.2.2 Local minimization from best random sample

Another strategy is to use the allotted time to (quasi)randomly sample the given search space and then run the local minimizer from the best sampled point.

1.3 Simulated annealing

Simulated annealing is a stochastic meta-heuristic algorithm for global minimization. The name and inspiration come from annealing—heating up and cooling slowly—in material science. The slow cooling allows a piece of material to reach a state with "lowest energy".

The objective function in the space of states is interpreted as some sort of potential energy and the points in the search space are interpreted as states of a certain physical system. The system attempts to make transitions from its current state to some randomly sampled nearest states with the goal to eventually reach the state with minimal energy – the global minimum.

The system is attached to a thermal reservoir with certain temperature T. Each time the energy of the system is measured the reservoir supplies it with a random amount of thermal energy sampled from the Boltzmann distribution,

$$P(E) = Te^{-E/T} \, . \tag{1}$$

Table 1: Simulated annealing algorithm

```
state ← start_state
T ← start_temperature
energy ← E(state)
REPEAT :
  new_state ← neighbour(state)
  new_energy ← E(new_state)
  IF new_energy < energy :
    state ← new_state
    energy ← new_energy
  ELSE :
    do with probability exp(−(new_energy−energy)/T) :
      state ← new_state
      energy ← new_energy
  reduce_temperature_according_to_schedule(T)
UNTIL terminated
```

If the temperature equals zero the system can only make transitions to the neighboring states with lower potential energy. In this case the algorithm turns merely into a local minimizer with random sampling.

If temperature is finite the system is able to climb up the ridges of the potential energy—about as high as the current temperature—and thus escape from local minima and hopefully eventually reach the global minimum.

One typically starts the simulation with some finite temperature on the order of the height of the typical hills of the potential energy surface, letting the system to wander almost unhindered around the landscape with a good chance to locate if not the best then at least a good enough minimum. The temperature is then slowly reduced following some annealing schedule which may be supplied by the user but must end with $T = 0$ towards the end of the allotted time budget.

Table 1 lists one possible variant of the algorithm. Here the function `neigbour` is system-dependent and should return a randomly chosen "neighbour" of the given state. For a continuous function, where the state is the position of the current approximation to the minimum, the neghbour could be a random position within radius R from the current position. The step-radius can be gradually reduced to zero toward the end of the simulation, like

$$R(t) = R_0 \cdot (1 - \frac{t}{t_a}) \,, \tag{2}$$

where R_0 is the initial radius, t is the running time, and t_a is the allotted time.

The temperature can be also reduced linearly,

$$T(t) = T_0 \cdot (1 - \frac{t}{t_a}) \,, \tag{3}$$

where T_0 is the initial temperature.

Table 2: Quantum annealing algorithm

```
state ← start_state
energy ← E(state)
R ← start_radius
REPEAT :
  new_state ← random_neighbour_within_radius(state,R)
  new_energy ← E(new_state)
  IF new_energy < energy :
    state ← new_state
    energy ← new_energy
  reduce_radius_according_to_schedule(R)
UNTIL terminated
```

1.4 Quantum annealing

Quantum annealing is a general global minimization algorithm which—like simulated annealing—also allows the search path to escape from local minima. However instead of the thermal jumps over the potential barriers quantum annealing allows the system to tunnel through the barriers.

In its simplest incarnation the quantum annealing algorithm allows the system to attempt transitions not only to the nearest states but also to distant states within certain "tunneling distance" from the current state. The transition is accepted only if it reduces the potential energy of the system.

At the beginning of the minimization procedure the tunneling distance is large—on the order of the size of the region where the global minimum is suspected to be located—allowing the system to explore the region. The tunneling distance is then slowly reduced according to a schedule such that by the end of the allotted time the tunneling distance reduces to zero at which point the system hopefully is in the state with minimal energy.

1.5 Evolutionary algorithms

Unlike annealing algorithms, which follow the motion of only one point in the search space, the evolutionary algorithms typically follow a set of points called a *population* of individuals. Somewhat like the downhill simplex method which follows the motion of a set of points – the simplex.

The population evolves toward more fit individuals where fitness is understood in the sense of minimizing the objective function. The parameters of the individuals (for example, the coordinates of the points in the parameter space of the objective function) are called genes.

The algorithm proceeds iteratively in discrete steps where the population in each iteration is called a generation. In each generation the fitness of each individual—typically, the value of the objective function—is evaluated and the new generation is generated stochastically from the gene pool of the current generation through

certain operations (like crossovers and mutations) such that the genes of more fit individuals have a better chance of propagating into the next generation.

Each new individual in the next generation can be produced from a pair of "parent" individuals of the current generation, as inspired by biology, but more than two "parents" can be used as well. The parents for a new individual are selected from the individuals of the current generation through a fitness based stochastic process where fitter individuals are more likely to be selected.

Generation of "children" continues until the population of the new generation reaches the appropriate size after which the iteration repeats itself.

The algorithm is terminated when the fitness level of the population is deemed sufficient or when the allocated budget is exhausted.

1.5.1 Particle Swarm Optimization (PSO)

One example of evolutionary optimization algorithms is the *particle swarm optimization* method [2] where a population of "particles" move (in discrete time-steps) through the parameter space of the objective function and sample the function at the particles' positions at each step. The movement of each particle is stochastic and is influenced by the particle's best position as well as the whole swarm's best position: at each time-step a particle gets a stochastic kick toward it's own best position and toward the swarm's best position. After a certain amount of steps the swarm is expected to converge to the best solution.

A particle number i carries three vector-parameters: its position $\mathbf{x}_i$ in the parameter space of the objective function; its best position $\mathbf{p}_i$ so far; and its velocity, $\mathbf{v}_i$. In addition the swarm as a whole remembers its global best position, $\mathbf{g}$.

At each time-step Δt (usually equal unity) first the positions of the particles are updated,

$$\mathbf{x}_i = \mathbf{x}_i + \mathbf{v}_i \Delta t \ , \tag{4}$$

and the objective function is sampled at the new positions. Then the particles' best and the global best positions are updated. After that the velocities of the particles are stochastically updated according to the formula,

$$\mathbf{v}_i = w\mathbf{v}_i + u(\mathbf{p}_i - \mathbf{x}_i) + u(\mathbf{g} - \mathbf{x}_i) \ , \tag{5}$$

where u is a random number from a unit uniform distribution, and $w < 1$ us the damping parameter which ensures that the swarm gradually calms down (hopefully in the area of global minimum).

Table 3 lists one possible implementation of the algorithm.

1.5.2 Bare bones PSO (BBPSO)

A simpler variant of the PSO algorithm is the the so called "bare bones PSO" [1]. Here one dispences with the velocity of the particles and instead updates the

Table 3: Particle swarm optimization algorithm

```
Initialize uniform unit random number generator u;
Assume the time–step is equal unity, Δt = 1;
Initialize particle positions randomly within given rectangular
volume V[a, b] given by the vectors a and b,
  x_i = random vector within V[a, b];
Initialize particle velocities randomly,
  v_i = random vector within V[(a−b)/2, (b−a)/2] 1/Δt;
Initialize local best positions,
  p_i = x_i;
Initialize global best position,
  g = min_i(f(p_i));
REPEAT:
  Update velocities (the damping parameter w ≈ 0.72),
    v_i = w v_i + U · (p_i − x_i) 1/Δt + U · (g − x_i) 1/Δt;
  Update positions,
    x_i = x_i + v_i Δt;
  Update local bests,
    if f(x_i) < f(p_i) p_i = x_i;
  Update global best,
    if f(x_i) < f(g) g = x_i;
UNTIL allotted time is spent or converged
```

positions of the particles using the following rule,

$$\mathbf{x}_i = \mathrm{G}\left(\frac{\mathbf{p}_i + \mathbf{g}}{2}, \|\mathbf{p}_i - \mathbf{g}\|\right) , \tag{6}$$

where $\mathbf{x}_i$, $\mathbf{p}_i$ are the position and the best position of particle i, $\mathbf{g}$ is the global best position, and $\mathrm{G}(\mathbf{x}, \sigma)$ is the Gaussian (normal) distribution with the mean $\mathbf{x}$ and standard deviation σ.

Generation of normally distributed sequences

Box-Muller transform A normally distributed sequence of numbers, r, with zero mean and unit variance (called standard normal distribution) can be constructed using the *Box-Muller transform* from two sequences, u_1 and u_2, which are uniformly distributed on the unit interval $(0, 1]$. The transformation is given as

$$r_1 = \sqrt{-2 \ln u_1} \cos(2\pi u_2) , \tag{7}$$

$$r_2 = \sqrt{-2 \ln u_1} \sin(2\pi u_2) . \tag{8}$$

The sequences r_1 and r_2 are independent random variables with a standard normal distribution.

A normal distribution R with a given mean m and variance σ can be constructed from the standard normal distribution r as

$$R = m + r\sigma \ . \tag{9}$$

Irwin-Hall distribution The size-n *Irwin-Hall distribution* is the distribution of the sum of n independent numbers, u_i, which are uniformly distributed on $(0, 1)$,

$$x = \sum_{i=1}^{n} u_i \ . \tag{10}$$

By the Central Limit Theorem as n increases the Irwin-Hall distribution $\mathrm{H}_n(x)$ approaches the normal distribution $\mathrm{G}_{\mu\sigma}(x)$ with the mean $\mu = n/2$ and the variance $\sigma^2 = n/12$,

$$\sqrt{\frac{n}{12}}\mathrm{H}_n\left(x\sqrt{\frac{n}{12}} + \frac{n}{2}\right) \overset{n\to\infty}{\longrightarrow} \mathrm{G}_{0,1}(x) \ . \tag{11}$$

This leads to a simple approximation where a standard normal distribution is given by the sum of 12 pseudorandom numbers on $(0, 1)$,

$$\sum_{i=1}^{12} u_i - 6 \approx \mathrm{G}_{0,1} \tag{12}$$

References

[1] James Kennedy. Bare bones particle swarms. *Proceedings of the 2003 IEEE Swarm Intelligence Symposium*, page 80–87, 2003.

[2] R. Poli. Analysis of the publications on the applications of particle swarm optimisation. *Journal of Artificial Evolution and Applications*, 2008:1–10, 2008.