Units, functions, etc

Minimization by Simulated Annealing

In simulated annealing we randomly generate solutions. If a solution is better than the existing solution, it's accepted. If it's worse than the current solution, it may be accepted, and that's what allows simulated annealing to avoid becoming stuck in local minima. The probability that a worse solution will be accepted depends on the "temperature": the higher the temperature, the higher the probability. The probability is given by:

$$\text{Prob}(New, Old, Temp) := \begin{cases} \text{return } 1 & \text{if } New < Old \\ \text{return } \exp\left[\dfrac{(Old - New)}{Temp}\right] & \text{otherwise} \end{cases}$$

If new solution is better (smaller) than the old solution, return 1, otherwise calculate an acceptance probability

It is desirable that at the start the temperature is high enough that the probability of selecting a worse solution is very high, because this allows the algorithm to "explore" the solution space. As the algorithm progresses the temperature is gradually lowered, and the solution settles into a minimum. The longer the annealing process, the more likely that the final minimum found will be the global minimum in the solution space.

We also need a function that will determine how a new candidate solution is chosen at each iteration. We will keep this outside the main function because the form of this function depends on the minimization problem that is to be solved. As an example we will define a function that is suitable for the minimization of a function with N arguments or a least squares fit with N arguments.

$$\text{RandomInts}(N, n) := \text{round}(\text{runif}(N, 0, n + 1) - 0.5)$$

Returns N random integers between 0 and n

$$\text{Candidate}(Values, MinValues, MaxValues, N) := \begin{cases} Indicies \leftarrow \text{RandomInts}(N, \text{last}(Values)) \\ \text{for } i \in Indicies \\ \quad Values_i \leftarrow MinValues_i + (MaxValues_i - MinValues_i) \cdot \text{rnd}(1) \\ Values \end{cases}$$

The function has the arguments:
Values: A vector with N rows, containing the current values of the arguments
MinValues: A vector with N rows, containing the minimum values of the arguments
MaxValues: A vector with N rows, containing the maximum values of the arguments
N: the number of arguments to randomly change

The simulated annealing function:

$$\text{Anneal}(Obj\_Function, MinValues, MaxValues, Temp, MinTemp, CoolRate, N, Ntries) := \begin{cases} LastSol \leftarrow \begin{cases} Rand \leftarrow \text{runif}(\text{rows}(MinValues), 0, 1) \\ MinValues + [(MaxValues - MinValues) \cdot Rand] \end{cases} \\ LastCost \leftarrow Obj\_Function(LastSol) \\ (BestSol \quad BestCost) \leftarrow (LastSol \quad LastCost) \\ \text{while } Temp > MinTemp \\ \quad \text{for } i \in 1 .. Ntries \\ \quad\quad CurrentSol \leftarrow \text{Candidate}(LastSol, MinValues, MaxValues, N) \\ \quad\quad CurrentCost \leftarrow Obj\_Function(CurrentSol) \\ \quad\quad P \leftarrow \text{Prob}(CurrentCost, LastCost, Temp) \\ \quad\quad (LastSol \quad LastCost) \leftarrow (CurrentSol \quad CurrentCost) \text{ if } P \geq \text{rnd}(\\ \quad\quad (BestSol \quad BestCost) \leftarrow (LastSol \quad LastCost) \text{ if } LastCost < Bes \\ \quad Temp \leftarrow Temp \cdot (1 - CoolRate) \\ \text{augment}(LastSol, BestSol) \end{cases}$$

The arguments are:
Obj_Function: the objective function to be minmized. The function must have one argument only, which is a vector of parameters that can be varied during the minimization.
MinValues: a vector with the minimum allowed values for the paraameters of the objective function.
MaxValues: a vector with the maximum allowed values for the paraameters of the objective function.
Temp: the starting temperature
MinTemp: the minium temperature. When this tempaerature is reached the function terminates
CoolRate: The cooling rate.
N: the number of parameters that can be changed each time a new candidate solution is created
Ntries: the number of times a new candidate solution will be created at each temperature

Note that unlike the gradient descent methods of minimization the algorithm does not start with guess values and then systematically move to the best solution it can find, regardless of how far that is from the guess values. It starts at a random point within a solution space that is bounded by MinValues and MaxValues, and searches within that space for better solutions. The algorithm terminates when the temperatrure reaches MinTemp, at which time it returns a two column matrix, with the current solution in column 1 and the best solution found in column 2. These will usually be very close, but if the annealing is too fast they may not be.

In general, each time a new candidate soluton is created it is desirable to only change some of the existing solution. This results in

the best parts of the existing solution being retained each time a candidate solution is accepted. N should therefore be set so that only some of the parameters ae varied each time a candidate solution is created (this is more important in som minimization problems than in others).

Example:

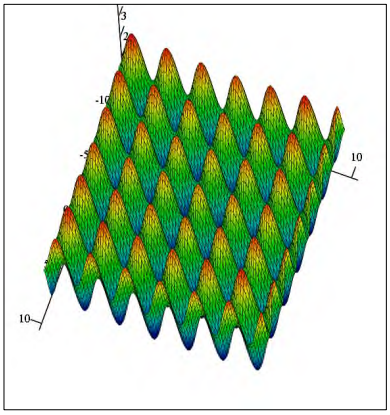
We wish to find the global minimum of this function:

$$f(\alpha,\beta,x,y) := \sin(2\cdot x) + \sin(2\cdot y) + \exp(\alpha\cdot|x|) + \exp(\beta\cdot|y|)$$

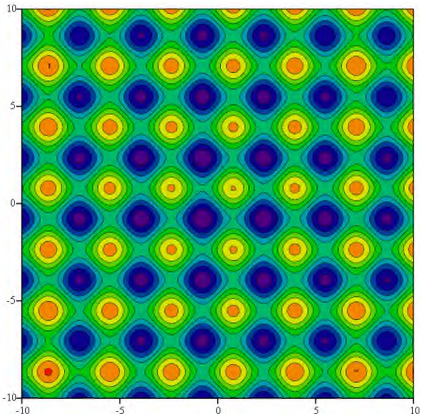
with $\alpha := 0.03$ $\beta := 0.03$

Plot the function over the range x=-10 to 10 and y=-10 to 10:

$$f2(x,y) := \begin{pmatrix} x \\ y \\ f(\alpha,\beta,x,y) \end{pmatrix} \qquad F := \text{CreateMesh}(f2,-10,10,-10,10,100,100)$$



F



F

The function clearly has very many local minima in the displayed range, and a gradient descent method would only find the global minimum if the guess values were ve close.

To apply the simulated annelaing algorithm we need to convert the form of the function to one with a single argument that is the vector of parameters, and create vectc of minimum and maximum values that define the extent of the search space:

$$f3(p) := f(\alpha,\beta,p_0,p_1)$$

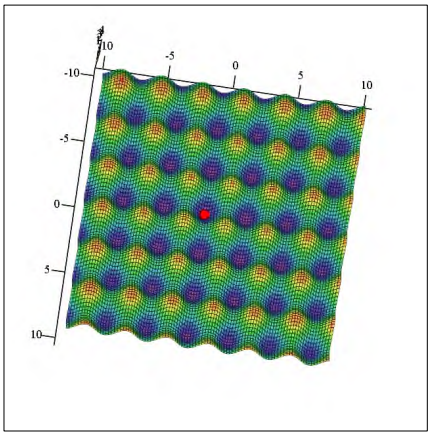
$$\text{MinVals} := \begin{pmatrix} -10 \\ -10 \end{pmatrix} \qquad \text{MaxVals} := \begin{pmatrix} 10 \\ 10 \end{pmatrix}$$

Now find the solution. We will wrp it in a small program that also calculates the time taken for convergence:

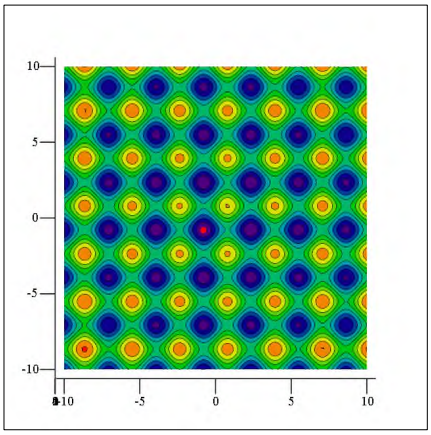
$$\begin{pmatrix} \text{Solution} \\ \text{Time} \end{pmatrix} := \left| \begin{array}{l} t0 \leftarrow \text{time}(0) \\ \text{Solution} \leftarrow \text{Anneal}(f3,\text{MinVals},\text{MaxVals},10,0.0001,0.01,1,10) \\ t1 \leftarrow \text{time}(0) \\ \begin{pmatrix} \text{Solution} \\ t1 - t0 \end{pmatrix} \end{array} \right.$$

$$\text{Solution} = \begin{pmatrix} -0.7754 & -0.7735 \\ -0.7854 & -0.7754 \end{pmatrix} \qquad f3\left(\text{Solution}^{\langle 0 \rangle}\right) = 0.0476 \qquad f3\left(\text{Solution}^{\langle 1 \rangle}\right) = 0.0475 \qquad \text{Time} = 0.065$$

$$X_0 := \text{Solution}_{0,0} \qquad Y_0 := \text{Solution}_{1,0} \qquad Z_0 := f3\left(\text{Solution}^{\langle 0 \rangle}\right)$$



F , (X , Y , Z + 0.2)



F , [X , Y , (0)]

In this case the simulated annealing is very good at finding the approximate location of the global minimum, but not very good at finding the exact minimum (at least, not in a reasonable time frame). However, we can refine the result using a gradient descent method:

p := Solution$^{\langle 0 \rangle}$ Solution := Minimize(f3 , p) Solution = $\begin{pmatrix} -0.7777 \\ -0.7777 \end{pmatrix}$ f3(Solution) = 0.0474

1)
tCost