Function Minimization (Continued):

There also exist other, more powerful Gradient function minimization methods.

One elegant and powerful one is the Variable Metric Method (VMM):

This method utilizes concepts from differential geometry, where an M-dimensional space is characterized by a metric $ds^2 = dx^T A dx$. The components of the $M \times M$ metric tensor $A$ transform in a covariant manner w.r.t. a change in coordinate systems, e.g. Cartesian/rectangular coordinates $(x, y, z)$ to spherical-polar coordinates $(r, \theta, \phi)$. {Please see Auxiliary P598AEM Lect. Notes on invariant/covariant/contravariant vectors/tensors if want/need further info on this…}.

In the VMM, we let the properties of the function $F(\lambda)$ (to be minimized) define a metric tensor. Assuming that $F(\lambda)$ is a scalar function, the second derivatives $\frac{\partial^2 F(\lambda)}{\partial \lambda_i \partial \lambda_j}$ are the components of a covariant tensor, so we define the metric tensor as: $(H_\lambda)_{ij} \equiv \frac{\partial^2 F(\lambda)}{\partial \lambda_i \partial \lambda_j}$.

Then $ds^2 = dx^T H_\lambda dx$ is an invariant quantity.

In the VMM, we let the properties of the function $F(\lambda)$ (to be minimized) define a metric tensor. Assuming that $F(\lambda)$ is a scalar function, the second derivatives $\frac{\partial^2 F(\lambda)}{\partial \lambda_i \partial \lambda_j}$ are the components of a covariant tensor, so we define the metric tensor as: $(H_\lambda)_{ij} \equiv \frac{\partial^2 F(\lambda)}{\partial \lambda_i \partial \lambda_j}$.

Then $ds^2 = dx^T H_\lambda dx$ is an invariant quantity.

In the VMM, we let the properties of the function $F(\lambda)$ (to be minimized) define a metric tensor. Assuming that $F(\lambda)$ is a scalar function, the second derivatives $\frac{\partial^2 F(\lambda)}{\partial \lambda_i \partial \lambda_j}$ are the components of a covariant tensor, so we define the metric tensor as: $(H_\lambda)_{ij} \equiv \frac{\partial^2 F(\lambda)}{\partial \lambda_i \partial \lambda_j}$.

Then $ds^2 = dx^T H_\lambda dx$ is an invariant quantity.

In the VMM, we let the properties of the function $F(\lambda)$ (to be minimized) define a metric tensor. Assuming that $F(\lambda)$ is a scalar function, the second derivatives $\frac{\partial^2 F(\lambda)}{\partial \lambda_i \partial \lambda_j}$ are the components of a covariant tensor, so we define the metric tensor as: $(H_\lambda)_{ij} \equiv \frac{\partial^2 F(\lambda)}{\partial \lambda_i \partial \lambda_j}$.

Then $ds^2 = dx^T H_\lambda dx$ is an invariant quantity.
If \( F(\lambda) \) were truly quadratic, then \( H_\lambda \) would be a constant \( \text{i.e. independent of } \lambda \) and we would be in a space with a “constant metric”. If \( F(\lambda) \) is not (terribly far from) quadratic, then we expect that \( H_\lambda \) will be a slowly varying function of \( \lambda \), in which case we have a variable metric.

In Newton’s Method, \( H_\lambda \) is recalculated at each step, typically a very time-consuming process. In the VMM, the covariance matrix \( V_\lambda = H^{-1}_\lambda \) is estimated in each step by taking the \( V_\lambda \) at the previous step and correcting it by applying information from the current step.

The “Matrix Updating Formula” (MUF) which does this job is different for different VMM’s. The MUF procedure is as follows:

a.) Pick starting values \( \lambda^0 \). Calculate the corresponding starting gradient \( g_\lambda^0 = \partial F(\lambda) / \partial \lambda \) and some approximation to the covariance matrix \( V_\lambda^0 \).

(n.b. \( V_\lambda^0 \) may be the full second derivative matrix, or simply the unit matrix \( \mathbb{1} \).)

b.) Take a step to:

\[
\lambda^1 = \lambda^0 - V_\lambda^0 g_\lambda^0.
\]

{Compare this to: \( \lambda = \lambda^0 - H_\lambda^{-1} g_\lambda \) in Newton’s Method, P598AEM Lect. Notes 23, p. 6}.

This would be the exact minimum if \( F(\lambda) \) were quadratic and if \( V_\lambda^0 \) were the true covariance matrix (and not an approximation). {n.b. In a more elaborate method, we might look for a minimum along the “line” from \( \lambda^0 \) to \( \lambda^1 \) and call that point “\( \lambda^1 \)”.}

Using \( \lambda^1 \) we calculate the new gradient \( g_\lambda^1 = \partial F(\lambda) / \partial \lambda \).

c.) Now we: • correct the covariance matrix \( V_\lambda \) using an updating formula:

\[
V_\lambda^1 = V_\lambda^0 + f \left( V_\lambda^0, \lambda^0, g_\lambda^0, \lambda^1, g_\lambda^1 \right)
\]

• replace \( g_\lambda^0 \) by \( g_\lambda^1 \), \( V_\lambda^0 \) by \( V_\lambda^1 \), \( \lambda^0 \) by \( \lambda^1 \)

and then repeat steps b.) and c.) until the process converges, within some specified tolerance.

One such function \( f \left( V_\lambda^0, \lambda^0, g_\lambda^0, \lambda^1, g_\lambda^1 \right) \), due to Davidon (Comput. J. 10, 406 (1968)) is:

\[
V_\lambda^1 = V_\lambda^0 + \frac{(\delta - V_\lambda^0 \gamma)^T (\delta - V_\lambda^0 \gamma)}{\gamma^T (\delta - V_\lambda^0 \gamma)}
\]

where: \( \delta = \lambda^1 - \lambda^0 \) and: \( \gamma = g_\lambda^1 - g_\lambda^0 \)

This version of \( f \) has some very nice properties, including the fact that if \( F(\lambda) \) is truly quadratic, then the approximate \( V_\lambda \) can be shown to → the true covariance matrix after a number of iterations.
The VMM technique works well and is much faster than Newton’s Method, since it does not have to recalculate $V_{\lambda}$ from scratch at each step. However, sooner or later computational round-off errors will accumulate, as will errors due to approximations. It then makes sense, at the end, to recalculate $V_{\lambda}$ from scratch in order to obtain computationally reliable error estimates on the $\lambda$-parameters, rather than use the approximation to the covariance matrix that has been building up all along.

Convergence can be tested after each step by looking at:

$$ r = V_{\lambda}^1 g_{\lambda}^1 $$

(which is the distance the $\lambda$-parameters will move in the next step, i.e. $\lambda^* - \lambda^1$)

or at:

$$ \frac{1}{2} \rho = \frac{1}{2} g_{\lambda}^T V_{\lambda} g_{\lambda}^1 $$, or at: $F(\lambda^1)$ itself.

We expect $F(\lambda^1) < F(\lambda^0)$ of course. At the minimum $r = 0$, since the gradient $g_{\lambda} \equiv \partial F(\lambda)/\partial \lambda = 0$ defines the minimum. (In “real” problems, $g_{\lambda}$ is never exactly $= 0$!)

Finally, $\frac{1}{2} \rho = \frac{1}{2} g_{\lambda}^T V_{\lambda} g_{\lambda}^1 = 0$ at a minimum. The value of $\frac{1}{2} \rho$ is usually used as the criterion for convergence, since it estimates how far $F(\lambda)$ is from its value at the minimum; when $\frac{1}{2} \rho$ is smaller than a preset limit, the process is stopped. (In “real” problems, $\frac{1}{2} \rho$ is never exactly $= 0$!)

Function Minimization with Constraints:

Until now, we have only discussed cases where no constraints are placed on the $\lambda$-parameters (e.g. no restrictions on their allowed ranges). Constraints on the $\lambda$-parameters can take many forms:

\[
\begin{align*}
\text{i.)} & \quad f_1(\lambda) = 0 \\
& \quad f_2(\lambda) = 0 \\
& \quad \vdots \\
\text{Constraint Equations}
\end{align*}
\]

\[
\begin{align*}
\text{ii.)} & \quad a_1 \leq \lambda_1 \leq b_1 \\
& \quad a_2 \leq \lambda_2 \leq b_2 \\
& \quad \vdots \\
\text{Simple Constant Limits}
\end{align*}
\]

\[
\begin{align*}
\text{iii.)} & \quad r_1(\lambda) \leq \lambda_1 \leq s_1(\lambda) \\
& \quad r_2(\lambda) \leq \lambda_2 \leq s_2(\lambda) \\
& \quad \vdots \\
\text{Simple Variable Limits}
\end{align*}
\]

\[
\begin{align*}
\text{iv.)} & \quad u_1(\lambda) \leq v_1(\lambda) \leq w_1(\lambda) \\
& \quad u_2(\lambda) \leq v_1(\lambda) \leq w_2(\lambda) \\
& \quad \vdots \\
\text{Implicit Variable Limits}
\end{align*}
\]

We shall consider techniques that modify $F(\lambda)$ in such a way that the ordinary unconstrained $\chi^2$ minimization methods will be applicable.
For **i.** **Constraint Equations**, we have already seen how “Lagrange Multipliers” \( \alpha \) can be introduced. Alternatively, we have also shown that if we can solve each equation for one of the \( \lambda \)-parameters explicitly, e.g. \( \lambda_i \), then we can **eliminate** that specific \( \lambda \)-parameter.

For **ii.** **Constant Limits**, a useful technique here is to **change variables**. The idea is to replace the \( M \lambda \)-parameters by \( M \eta \)-parameters, where the \( \eta \)-parameters are functions of the \( \lambda \)-parameters, i.e. \( \eta = \eta(\lambda) \) with the property that when \( \lambda_k \) takes on values from \( a_k \) to \( b_k \), then \( \eta_k \) takes on values from \( -\infty \) to \( +\infty \). The function \( \eta(\lambda) \) should **not** introduce any new minima in \( \chi^2 \) and should also be “smooth” so that it does not severely **distort** or add any **weird features** to the existing problem.

Of course, once the problem has been solved in \( \eta \) space, the **covariance matrix** \( V_{\eta \lambda} \) of the \( \eta \)-parameters (as well as the \( \eta \)-parameters themselves) must be transformed back to \( \lambda \) space.

**Examples:**

To limit \( 0 \leq \lambda_k \leq \infty \), e.g. choose: \( \lambda_k = \eta_k^2 \) or: \( \lambda_k = e^{\eta_k} \)

To limit \( 0 \leq \lambda_k \leq 1 \), e.g. choose: \( \lambda_k = \sin^2 \eta_k \) or: \( \lambda_k = \frac{e^{\eta_k}}{e^{\eta_k} - e^{-\eta_k}} \)

To limit \( a_k \leq \lambda_k \leq b_k \) e.g. choose: \( \lambda_k = a_k + b_k \sin^2 \eta_k \), etc.

Note that the \( \pi \)-periodicity of the \( \sin^2 \eta_k \) function implies that a particular value of \( \lambda_k^* \) corresponds to **many** \( \eta_k^* \), each differing by \( \pi \) from each other. This means that while searching in \( \eta_k \) space to minimize \( \chi^2 \), the \( \eta_k \)-steps should be small enough so that the search doesn’t **ever** jump from the neighborhood of \( \eta_k \) to \( \eta_k \pm \pi \), or else the procedure will become **very** confused...

This procedure can be extended in many ways to more complex constraints. For example, to minimize \( fcn(\lambda_1, \lambda_2) \) with the requirement that \( 0 \leq \lambda_1 \leq \lambda_2 \leq \infty \) we could define \( \lambda_1 = \eta_1^2 \), \( \lambda_2 = \eta_1^2 + \eta_2^2 \). **This enforces the above requirement.**

For **iii.** **Simple Limits** and **iv.** **Variable Limits**, a useful technique employs a “**Penalty Function**”. Instead of transforming the **variables** in \( F(\lambda) \) as we did above, we modify \( F(\lambda) \) itself in such a way that it is **forced** to become very large where we **don’t** want \( \lambda \) to go, but is unaffected for values of \( \lambda \) that **are** allowed.

For example, in order to **minimize** \( fcn(\lambda) \) with the **additional constraint** \( p(\lambda) \geq 0 \), we can define a “penalty function” \( T(\lambda) \) as:

\[
T(\lambda) = \begin{cases} 
0 & \text{if } p(\lambda) \geq 0 \\
\epsilon[p(\lambda)]^2 & \text{if } p(\lambda) < 0 
\end{cases}
\]

where \( \epsilon \) is some **“large” positive** number, i.e. **large** compared to **any** value of \( F(\lambda) \) in the region being searched near the **minimum** of \( F(\lambda) \).
We then minimize the function $F'(\hat{\lambda}) = F(\hat{\lambda}) + T(\hat{\lambda})$, i.e. we minimize:

$$F'(\hat{\lambda}) = \begin{cases} F(\hat{\lambda}) & \text{if } p(\hat{\lambda}) \geq 0 \\ F(\hat{\lambda}) + c[p(\hat{\lambda})]^2 & \text{if } p(\hat{\lambda}) < 0 \end{cases}$$

Thus, $F'(\hat{\lambda})$ will be continuous at the “boundary” of the constraint, i.e. where $p(\hat{\lambda}) = 0$.

And, as soon as $\hat{\lambda}$ “wanders” into a region where $p(\hat{\lambda}) < 0$, $F'(\hat{\lambda})$ gets large, and so $\hat{\lambda}$ will wander right back out!

**Finding Multiple Minima and the Global Minimum:**

Although the physical problem that motivates finding a minimum is assumed to have “a minimum point”, i.e. a “best” solution, the function we use to solve the problem may in fact have several (local) minima. Ideally, a “solution” should find all of the minima, leaving us free to think about the results and choose the one that is the “best” answer to our problem. Any of the techniques that we have discussed so far will (at best) find only one minimum, which may or may not be the Global minimum, and may not even be the one closest to our starting point.

For physical problems we often assume that the minimum we want is the one nearest the starting values of some (or all) of the parameters (i.e. we have foreknowledge of, or approximate values for one or more of them.) It is not even common practice to demand that the minimum we want be the **Global** minimum. It is rare to ask for, and virtually impossible to find, all of the minima of a complicated function of many parameters if only a limited amount of computer time is available. Unfortunately, no methods in common use today are guaranteed to work for finding several minima.

In principle, a “grid search” could be made in which a grid of starting values $\hat{\lambda}^0$ was used. For each starting value, a complete minimization could be done. If the grid were fine enough, then we’d probably find all of the minima of $F(\hat{\lambda})$ in the region searched. But the “cost” would be high!

One practical possibility is to do a Monte Carlo search once $\hat{\lambda}$ has been found. We look at $F(\hat{\lambda})$ at many other points and see whether $F(\hat{\lambda}) < F(\hat{\lambda}^*)$ anywhere, which would prove that $\hat{\lambda}$ was not the **Global** minimum. A new minimization could then begin at that $\hat{\lambda}$. This will eventually find deeper and deeper minima (if any), but will not find any shallower ones, if these exist/are present.
Some analytic tools have been developed for finding other minima. For example, since the first derivatives \( \frac{\partial F(\lambda)}{\partial \lambda} \bigg|_{\lambda = \lambda^*} \) vanish at the minimum of the function \( F(\lambda) \), at \( \lambda = \lambda^* \), then near the minimum, the Taylor series expansion is:

\[
F(\lambda) = F(\lambda^*) + \frac{1}{2} (\lambda - \lambda^*)^T H_{\lambda} (\lambda - \lambda^*) + \text{higher-order terms}
\]

Now if other minima are present in \( F(\lambda) \), the higher order terms in the Taylor series expansion will be important, since information about the other minima must be contained in these higher-order terms.

The trick, then is to transform \( F(\lambda) \) such that it has no minimum at \( \lambda = \lambda^* \), but it still retains the information contained in the higher-order terms – e.g. define:

\[
F_1(\lambda^*, \lambda) = \frac{2[F(\lambda) - F(\lambda^*)]}{(\lambda - \lambda^*)^T H_{\lambda} (\lambda - \lambda^*)}
\]

Near \( \lambda^* \), \( F_1(\lambda^*, \lambda) \rightarrow 1 + \text{higher-order terms} \), so it (presumably) has no minimum at \( \lambda = \lambda^* \).

Then we can look for a minimum/look for minima associated with \( F_1(\lambda^*, \lambda) \)....

Methods such as this have been found to work in some cases, but their general validity has not yet been established...

The Metropolis-Hastings Algorithm and Simulated Annealing:

The Metropolis-Hastings algorithm [N. Metropolis, et al., J. Chem. Phys. 21 (6): 1087-92 (1953); W.K. Hastings, Biometrika 57 (1): 97-109 (1970] is a Markov chain (i.e. random-walk) Monte Carlo algorithm used for obtaining a sequence of random samples from a function \( F(\lambda) \) for which direct sampling may be difficult. It was first proposed for use with the Boltzmann distribution \( f(E; T) \propto e^{-E(T)/k_b T} \). The M-H random walk-type algorithm can be helpful, when used in combination with a function minimization algorithm, such as the Simplex Method, to efficiently do so, and hopefully find the global minimum of \( F(\lambda) \).

The sequence of random samples of \( F(\lambda) \) can also be used e.g. to approximate the function \( F(\lambda) \) or to compute an integral of the function \( F(\lambda) \) – i.e. a C.D.F., or an expectation value, ...

The M-H algorithm has been used to solve the famous Traveling Salesman Problem – minimizing the total round-trip path for visiting a sequence of cities only once {there are even websites where you yourself can solve your own trip, e.g. http://www.gebweb.net/optimap/}. The M-H algorithm has also been used for VLSI chip design, PC board layout – routing traces to minimize their length, using e.g. Lagrange multipliers in order to minimize the # PC board jumpers, etc. The M-H algorithm has also found use(s) in calculating many-body spin configurations, DNA & genome sequencing...
The strategy employed by coupling the Metropolis-Hastings algorithm e.g. to the Simplex function minimization method is inspired by the microscopic thermodynamical physics associated with the slow (i.e. adiabatic) cooling of a heated metal – i.e. the annealing of the metal.

At high temperatures, the number of local microstates available to be “explored” by an atom of the metal is large – due to the high local energy density – which is proportional to the temperature $T$ of the metal. The atoms of the metal would all like to be in their lowest possible energy states (thereby maximizing the overall entropy of the system) but sometimes, due to temporal and spatial fluctuations in the (local) thermal energy density, individual atoms may (momentarily) find themselves in a higher energy state.

As the metal slowly cools, the number of local microstates available to be “explored” by an atom slowly decreases due to the decrease in thermal energy; the thermal fluctuations are also reduced.

For an adiabatically slow cooling process, the atoms in the metal eventually find their lowest energy state, forming a giant single crystal if no defects are present, at the end of the cooling process.

At $T = 0$, the “internal energy” $E$ associated with the (scalar) function $F(\lambda)$ to be minimized is the value of the function itself, i.e. $E(T = 0) = F(\lambda)$, at the $\{M\}$-dimensional “space” point $\lambda$.

The simulated annealing procedure of the Metropolis-Hastings + Simplex method starts off at a “high” temperature $T_{max}$ {which hopefully is $T_{max} > E_{max}/k_B = F_{max}(\lambda)/k_B$ }. Thus, at finite temperature $T$, the “metal” {the ensemble of {continuum} values of the {spatial} $\lambda$-parameters associated with the (scalar) function $F(\lambda)$ } must have an average thermal energy density $\Delta u_{th}(T)$ and also have thermal fluctuations in the local energy density $\delta u_{th}(T, \lambda)$ associated with it.

Recall that the simplex function minimization replaces the $M$-dimensional “space” point $\lambda$ by an $M+1$ dimensional simplex – e.g. a triangle in 2-D or a tetrahedron in 3-D, etc.

We thus add a positive, logarithmically-distributed random variable, proportional to the temperature $T$, to the stored function value $F(\lambda)$ associated with every vertex of the simplex:

$$E(T, \lambda) = F(\lambda) + \Delta U_{th}(T) + \delta U_{th}(T)$$

Note that since the thermal contribution $\Delta U_{th}(T) + \delta U_{th}(T) = Vol \times [\Delta u_{th}(T) + \delta u_{th}(T)]$ to the internal energy is logarithmically-distributed, then the corresponding changes in the (local) Maxwell-Boltzmann probability density function $f(E; T) \propto e^{-E/T}/k_B T$ will be linear.

In analogy to the annealing process of a real metal, if the simulated thermodynamical system is offered a succession of options, we assume that it will (on average) change its configuration from energy $E_1 \equiv E(T, \lambda_1)$ to $E_2 \equiv E(T, \lambda_2)$ with probability $p = e^{-\Delta E/k_B T} = e^{-(E_2 - E_1)/k_B T}$. 


However, note that if $E_2 < E_1$, then $p = e^{-(E_2 - E_1)/k_B T} > 1$. In order to prevent the overall function minimization process of this algorithm from being “too greedy” – i.e. finding the minimum of the function $F(\lambda)$ too rapidly – thus “quenching” the metal instead of “annealing” it {and thus increasing the probability of the algorithm getting “trapped” in a local minimum}, the M-H algorithm prevents $p = e^{-(E_2 - E_1)/k_B T}$ from exceeding 1 by clamping $p = 1$.

This information is then passed to the simplex, informing it that going from $E_1 \equiv E(\lambda_1)$ at $\lambda_1$ to $E_2 \equiv E(\lambda_2)$ at $\lambda_2$ is a good choice, but not (way) too good of a choice... The simplex will thus always accept this $p = 1$ move.

The $M+1$ dimensional simplex then does its work moving around in the $M$-dimensional space of the $\lambda$-parameters, reflecting, expanding and/or contracting as it evaluates the function $F(\lambda)$ at the current $M+1$ vertex points of the simplex.

However, for every new $\lambda$-point that the simplex tries, the M-H algorithm subtracts a logarithmically-distributed random variable $\delta E_{th}(T)$, proportional to the temperature $T$, from the stored energy value $E(T, \lambda)$ associated with that new $\lambda$-vertex-point of the simplex.

Since $p = 1$ for $E_2 < E_1$, then $p \leq 1$ for $E_2 \geq E_1$, which will sometimes encourage the simplex to investigate/probe the $\lambda$-parameter space where fluctuations in the local thermal energy density have somewhat disfavored it.

Thus, because of the subtraction of thermal energy $\delta U_{th}(T)$ to each new point of the simplex, in order to encourage the simplex to continue with its function-minimization mission, the temperature $T$ must be (slowly/adiabatically) reduced after a period of allowing the simplex to do its minimum-finding thing for a while – this temperature reduction is known as the annealing, or cooling schedule for this algorithm.

Note that the “size” of the simplex in $\lambda$-parameter space is proportional to the temperature $T$. At “high” temperatures, the simplex expands to a “big” scale, able to “freely” sample/range over much of the $\lambda$-parameter space. It then executes a stochastic, tumbling-type of Brownian motion within that region, sampling new, approximately random points as it does so. The efficiency with which a region of the $\lambda$-parameter space is explored is independent of its narrowness and its orientation. If the temperature $T$ is reduced sufficiently slowly (i.e. allowing the “metal” to anneal), the size of the simplex also “shrinks” commensurately, enabling it to fit into/sample various local minima and thus also give it a good chance of finding the Global minimum of the function $F(\lambda)$.

As one might anticipate, some non-trivial optimization of parameter(s) is required for the M-H + simplex algorithms in order to successfully find the Global minimum of the function $F(\lambda)$. 

---

P598AEM Lecture Notes 24
Here again, a “one-size-fits-all” approach will not be optimal for minimizing all functions \( F(\lambda) \).

- The **starting temperature** \( T_{\text{max relative}} \) to the function \( F(\lambda) \) must be determined/optimized, *i.e.* the numerical value of Boltzmann’s constant \( k_b \) must be “tuned” since it is the proportionality constant between \( E \) and \( T \): \( E = k_b T \).

- The **scale** of the (mean) thermal energy \( \Delta U_{th}(T) \) and (logarithmic) random fluctuations in thermal energy \( \delta U_{th}(T) \) relative to the **dynamic range** of numerical value(s) of the function \( F(\lambda) \) must be determined/optimized, **and** the commensurate annealing/cooling schedule must also be determined/optimized.

- The **temperature dependence** of the simplex step size \( \delta \lambda_{\text{anneal}}(T) \) must also be optimized.

There likely will be detailed issues with “sufficiently slowly” cooling/annealing of the “metal” and the optimization of the annealing/cooling schedule, as success and/or failure of this function minimization is quite strongly dependent on the details of the annealing/cooling schedule.

Some algorithmic annealing/cooling schedule possibilities to try here are:

- Reduce the temperature \( T \) to \((1-e)T\) every \( N \) moves of the simplex, where \( e/N \) is empirically-determined – *i.e.* optimize \( e/N \) by trial & error…

- Budget a total of \( N_{\text{tot}} \) moves of the simplex, and reduce \( T \) after every \( N \) moves to a value \( T = T_{\text{max}} \left(1 - N_{\text{cum}}/N_{\text{tot}} \right)^\alpha \) where \( N_{\text{cum}} \) is the cumulative # simplex moves thus far and the exponent \( \alpha \) is a constant ranging from \( 1 \leq \alpha \leq 4 \). Thus, the parameters \( N_{\text{cum}} \) and \( \alpha \) need to be determined/optimized.

- After a set of \( N \) moves, set \( T = \beta \left( F(\lambda) - F(\lambda_{\text{best}}) \right) \) where \( \beta \) is an empirically-determined (i.e. optimized) constant of \( O\{1\} \) and \( F(\lambda) \) is the smallest current value of the simplex, whereas \( F(\lambda_{\text{best}}) \) is the best-ever encountered value of the simplex. However, *never* allow the temperature \( T \) to be reduced by more than some small fraction \( \gamma \leq 1 \) at a time.

Another (sometimes) useful thing to do is to allow an occasional complete restart of the M-H + Simplex function minimization algorithm {making certain that the best-ever encountered value of the simplex is not currently in the restart of the simplex!}.

For some, but not all function minimization problems, when the temperature \( T \) has been reduced by a factor of \( \sim 3 \times \) from its initial starting value, a restart can be beneficial/helpful.

The figure below shows the results of three random-walk Markov chains (yellow, green and blue) running on the 3-D Rosenbrock function

\[
F(x, y, z) = (1-x)^2 + (1-y)^2 + 100(y-x^2)^2 + 100(z-y^2)^2
\]

using the Metropolis-Hastings algorithm.
The M-H algorithm samples regions of \( F(x, y, z) \) and the Markov chains begin to mix in these regions. The approximate position of the minimum has been illuminated – as white light. Note that the red points are the ones that remain after a “burn-in” process. The earlier ones (yellow, green and blue) have been discarded.