Optimization Challenge Problems
(2-D and Single OF)
R. Russell Rhinehart

Challenges for Optimizers:

This set of objective functions (OF) was created to provide sample challenges for testing optimizers. The examples are all two-dimensional, having two decision variables (DV) so as to provide visual understanding of the issues that they embody. Most are relatively simple to program and compute rather simply, for user convenience. Most represent physically meaningful situations, for the person who wants to see utility and relevance. All are presented with minimization as the objective. All DVs and OF values are scaled on a 0 to 10 basis for common presentation.

Classic challenges to optimizers are objective functions that have:

1. non-quadratic behavior,
2. multiple optima,
3. stochastic responses,
4. asymptotic approach to optima at infinity,
5. hard inequality constraints, or infeasible regions,
6. slope discontinuities (sharp valleys),
7. a gently sagging channel (effectively slope discontinuities),
8. level discontinuities (cliffs),
9. flat spots,
10. nearly flat spots,
11. very thin global optimum in a large surface, pin-hole optima, improbable to find,
12. discrete, integer, or class DVs mixed with continuous variables,
13. underspecified problems with infinite number of equal solutions,
14. discontinuous response to seemingly continuous DVs because of discretization in a numerical integration, and
15. Sensitivity of DV or OF solution to givens.

In all equations that follow, x1 and x2 are the DVs, and f_of_x is the OF value. The DVs are programmed for the range [0, 10]. However, not all functions use DV values in that range. So, the DVs are scaled for the appropriate range and labeled x11 and x22. The OF value f_of_x is similarly scaled for a [0, 10] range. Any solution depends on the optimizer algorithm, the coefficients of the algorithm, and the convergence criteria. For instance, a multi-player optimizer has an increased chance of finding the global optimum. An optimizer based on a quadratic surface assumption (such as successive quadratic or Newton's) will jump to the optimum when near it, but can jump in the wrong direction when not in the proximity. The values for optimizer coefficients (scaling, switching, number of players, number of replicates, initial step size, tempering or acceleration) can make an optimizer efficient for one application, but with the same values it might be sluggish or divergent in an application with other features. The convergence criteria may be right for one optimizer, but stop another long before arriving at an optimum. When you are exploring optimizers, realize that the results are dependent on your choice of optimizer coefficients and convergence criteria, as well as the optimizer and the features of the test function.

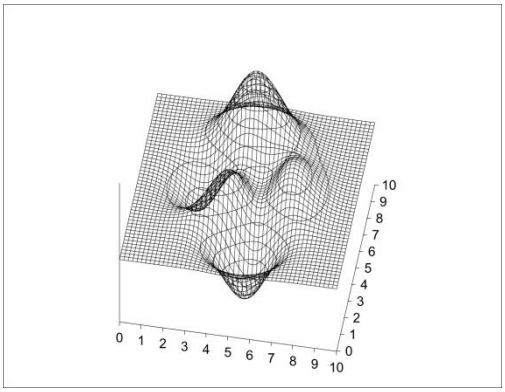
There are several metrics used to assess optimizer (and coefficient choice) performance. We desire to have:

1. A minimum number of function evaluations (NOFE). Each function evaluation represents either computer work, or experimental cost. Be sure to include either numerical or analytical evaluations of gradients and Hessian elements in the NOFE. These derivative evaluations might be a part of the optimizer, or they could be within a convergence criterion. Count them all.
2. Minimum computer time. Nominally, the OF evaluation is the most time consuming aspect, but where the computer must sort through multiple players or apply convoluted logic, the optimizer logic may take more time than the OF evaluation. When you accumulate optimizer time, be sure to exclude unnecessary display I/O and the function evaluation time.
3. Maximum probability of finding the global optima. When there are multiple optima, traps, diversions to a constraint, etc. the likelihood of any particular run of an optimizer in finding the global is important. You many need 1000 or more runs from random initializations to be able to assess the probability of an optimizer to find the global. Accuracy could be measured by the probability that the optimizer identifies the global (converges in the vicinity of the global).
4. Precision. This is the closeness of the optimizer to the true optimum. The optimizers are numerical procedures which also have finite convergence stopping criteria. They will not stop exactly at the true optimum. Closeness to the optimum can be assessed either by the OF value or by the DV value deviations from the true optimum. We talk about finding the global optimum, but the reality is that the optimizer finds the proximity of an optima, not the exact point. Precision could be measured by the rms (root-mean-square) deviation (either DV from DV*, or OF from OF*) from those trials that located the global.
5. Robustness. This is a measure of the optimizer ability to cope with surface aberrations (cliffs, flat spots, slope discontinuities, hard constraints, stochastic OF values, a trial solution that is infeasible or cannot return an OF value). It also includes the optimizer ability to generate a next feasible trial solution regardless of the surface features. Perhaps a measure of robustness could be the fraction of times the optimizer can generate a feasible next trial solution.
6. Scalability. As the number of DVs increases, how does the computational time or storage increase? How does the NOFE increase? How do the requirements on the user (such as the number of user-chosen coefficient values that need to be specified for either the optimizer or the convergence criteria) increase? The burden might be acceptable for low order applications, but excessive for higher dimension ones. Do the diverse aspects rise linearly with DV dimension, as a quadratic, or exponentially?
7. User understanding. How easy is it for the user to understand the optimizer logic and computations? How easy is it for the user to establish confidence that the optimizer result is in desired proximity of the global? User ability to adapt the code. Algorithm complexity.

Since any of these metrics will depend on the initial trial solution(s) and the surface features of a specific objective function, you will need to run many trials and calculate an average value representing individual functions. However, replicate trials from random initializations will not produce exact duplicate results. For simple problems, perhaps 20 trials are fully adequate to have relatively certain values of the statistics. However, you may need 100 to 10,000 trials to determine representative values for probability statistics associated with other attributes. You should keep running trials until the standard deviation of the statistic of interest is small enough to confidently differentiate between statistic values representing different experimental conditions (optimizers, coefficients, convergence criteria). Statistical comparisons need to be made on replicate results, such as a t-test of differences in NOFE. The Central Limit Theorem reveals that the standard error of the average is inversely proportional to the square root of the number of replicate trials.

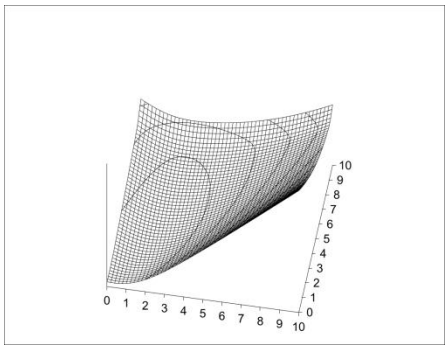
Further, one optimizer (Newton's, successive quadratic) might jump to the solution in one step if the function is a simple quadratic response, but it might become hopelessly lost on a different function. How do you compare optimizers? You need to choose a set of test functions that represent a diversity of features.

Peaks (#30) – This function seems to be a MatLAB creation as a simple representation of several optimization issues. It provides mountains and valleys in the middle of a generally outward sloping surface. There are two major valleys, and between the mountains, a small local minima. If the trial solution starts on the North or East side of the mountains, it leads downhill to the North or East boundary. The Southern valley has the lowest elevation. The surface is non-quadratic, and has multiple optima.
The function is:




```
x11 = 3 * (x1 - 5) / 5      'convert my 0-10 DVs to the -3 to +3 range for the function
x22 = 3 * (x2 - 5) / 5
f_of_x = 3 * ((1 - x11) ^ 2) * Exp(-1 * x11 ^ 2 - (x22 + 1) ^ 2) - _
        10 * (x11 / 5 - x11 ^ 3 - x22 ^ 5) * Exp(-1 * x11 ^ 2 - x22 ^ 2) - _
        (Exp(-1 * (x11 + 1) ^ 2 - x22 ^ 2)) / 3
f_of_x = (f_of_x + 6.75) / 1.5      'to convert to a 0 to 10 f_of_x range
```

Shortest Time (#47) – This simple appearing function is very confounding for successive quadratic or Newton's approaches. It represents a simple situation of a person on land on one side of a shallow river wanting to minimize the travel time to a point on land on the other side. It is also kin to light traveling the minimum time path through three media. Variables v1, v3, and v3 are the velocities through the near side, water, and far side; and x1 and x2 represent the E-W distance that the path intersects the near side and far side of the river.




```
v1 = 1
v2 = 0.75
v3 = 1.25
xa = 1
ya = 1
xb = 9
yb = 9
a1 = 3
b1 = -0.3
a2 = 5
b2 = 0.4
y1 = a1 + b1 * x1
y2 = a2 + b2 * x2
distance1 = Sqr((x1 - xa) ^ 2 + (y1 - ya) ^ 2)
distance2 = Sqr((x2 - x1) ^ 2 + (y2 - y1) ^ 2)
```

```
distance3 = Sqr((xb - x2) ^ 2 + (yb - y2) ^ 2)
timetravel = distance1 / v1 + distance2 / v2 + distance3 / v3
f_of_x = (timetravel - 12) * 10 / (32 - 12)
```

The problem statement from 2012 was: "She was playing in the sandy area across the river from her house when the dinner bell rang. To get home she needed to run across the sand, run through the shallow lazy river, then run across the field to home. She runs faster on land than on sand. Both are faster than running though the knee-deep water. What is the shortest-time path home? On the x-y space of her world, she starts at location (1,1) and home is at (9,9). Perhaps the units are deci-kilometers (dKm) (tenths of a kilometer). Her speed through water is 0.75 dKm/min, through sand is 1.0 dKm/min, and on land is 1.25 dKm/min. The boundaries of the river are defined by y = 3.0 - 0.3x and y = 5.0 + 0.4x."

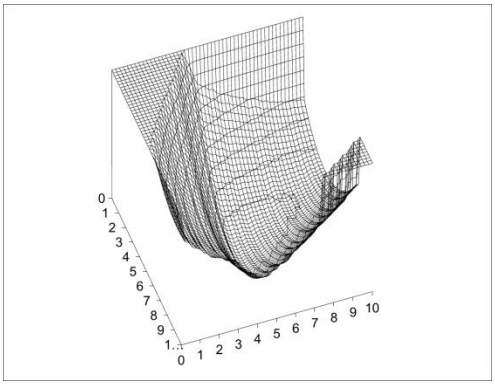
Hose Winder (#46) – This function presents discontinuities to the generally well-behaved floor. It represents a storage box that winds up a garden hose on a spool. When the winding hose gets to the end of the spool, it starts back but at a spool diameter that is larger by two hose diameters. The diameter jump causes the discontinuities. The objective is to design the storage box size and handle length to minimize the owner's work in winding the hose.

The 2012 problem statement was: "Consider that a hose is 200 feet long, and 1.25 inch in diameter, and is wound on a 6 inch diameter spindle by a gear connection to the handle. The hose goes through a guide, which oscillates side-to-side to make the winding uniform. Each sweep of the guide leads to a new hose layer, making the wind-on diameter 2.5 inches larger.

Originally the hose is stretched out 200 ft. To reel it in, the human must overcome the drag force of the hose on the ground. Either a smaller spindle diameter or a larger handle radius reduces the handle force required to reel in the hose. As the hose is reeled in, its residual length is less, and the drag force is less. But, when the first spindle layer is full and the hose moves to the next layer, the leverage changes, and the wind-up handle force jumps up.
After winding 200 ft. of hose, the human is exhausted. He had to overcome the internal friction of the device, the hose drag resistance, and move his own body up and down. We wish to design a device that that minimizes the total human work (handle force times distance moving the handle + body work). We also wish to keep the maximum force on the handle less than some excessive value (so that an old man can do the winding). If you increase the handle radius, the force is lessened, but the larger range of motion means more body motion work. There is a constraint on the handle radius – it cannot make the human's knuckles scrape the ground.

If you make the spindle length longer, so that more hose is wound on each layer, then the hose wind-on diameter does not get as large, and the handle needs less force to counter the drag. But, more turns to wind-in the hose means more body motion.

Further, consider the economics of manufacturing the box. The box volume needs to be large enough to windup the hose on the spindle, and perhaps 20% larger (so that spiders can find space for their webs, I think). Use 5 cuft. If the side is square, then defining the length and volume sets the side dimensions. Setting the weight and cost directly proportional to the surface area, the spindle length defines the cost of manufacturing and shipping. The objective function is comprised of a penalty for the maximum force, a penalty for the cost, and a penalty for the wind-up work."



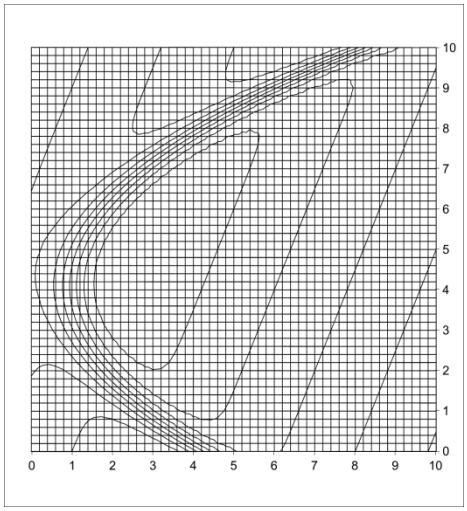
The 3-D view indicates a generally smooth approach to a minimum, but with local wrinkles on the surface. The local valleys guide many optimizers to a false minimum.

The VBA code is:

```
If x1 <= 0 Or x2 <= 0 Then
    constraint = "FAIL"
    f_of_x = 15
    Exit Function
End If
'   HandleRadius = x1 / 4            'nominal, scales 0-10 to 0-2.5 ft
'   BoxWidth = x2 / 2                'nominal, scales 0-10 to 0-5 ft
HandleRadius = 1.8 + 0.04 * x1      'to focus on discontinuties'
BoxWidth = 0.6 + 0.02 * x2          'to focus on discontinuities
WindRadius = 0.25                   'spindle radius 3 inches as 1/4 ft
BoxVolume = 5                       'cuft
BoxSide = Sqr(BoxVolume / BoxWidth)
If HandleRadius > 0.85 * BoxSide Then
    constraint = "FAIL"
    f_of_x = 15
    Exit Function
End If
Area = 2 * BoxSide ^ 2 + 3 * BoxWidth * BoxSide     'no bottom closure
HoseLength = 200                    'ft
HoseDiameter = 0.1                  '1.25 inches = 0.1 ft
DragLength = HoseLength
dcircle = 0.025                     'fraction of circumfrence
work = 0
WindLength = 0
Fmax = 0
Do Until DragLength < 2             'enough increments to wind up hose, leave 2 feet of hose outside
winder
    DragForce = 0.2 * DragLength        '0.2 is coefficient of friction lbf/ft of hose
    HandleForce = (DragForce * WindRadius + 0.5) / HandleRadius + 1  'hose drag, torque to spin
assembly, body motion
    If HandleForce > Fmax Then Fmax = HandleForce
    dwork = HandleForce * HandleRadius * 2 * 3.14159 * dcircle
    work = work + dwork
```

```
    dlength = WindRadius * 2 * 3.14159 * dcircle     'incremental length wound in one circle increment
    WindLength = WindLength + dlength              'total length wound on the layer
    DragLength = DragLength - dlength              'length of hose left unwoound
    If WindLength > (BoxWidth / HoseDiameter) * 2 * 3.14159 * WindRadius Then    'layer full, move to next
      WindRadius = WindRadius + HoseDiameter      'update winding radius
      WindLength = 0                              'reset wound length on new layer
    End If
Loop
'   f_of_x = 10 * ((Fmax / 5) ^ 2 + (Area / 20) ^ 2 + (work / 1000) ^ 2 - 28) / (70 - 28)  'nominal
f_of_x = 10 * ((Fmax / 5) ^ 2 + (Area / 20) ^ 2 + (work / 1000) ^ 2 - 28.25) / (28.45 - 28.25) 'to focus on discontinuities
```

Boot Print in the Snow (#19) – This represents a water reservoir design problem, but with very simple equations. The objectives of a reservoir are to trap excessive rain water to prevent downstream floods, to release water downstream to compensate for upstream droughts, and to provide water for human recreational and security needs. We also want to minimize the cost of the dam and land. The questions are how tall should the dam be and how full should the reservoir be kept. The taller it is, the more it costs; but the lower will be the probability of flood or drought impact, and the better the recreational and security features. The fuller it is kept the less it can absorb floods, but the better the drought or recreational performance. On the other hand if kept nearly empty it can mitigate any flood, but cannot provide recreation or drought protection. The contour appears as a boot print in the snow. The contours represent the economic risk (probability of an event times the cost of the event). The minimum is at the toe. The horizontal axis, x1, represents the dam height. The vertical axis, x2, is the setpoint portion of full.

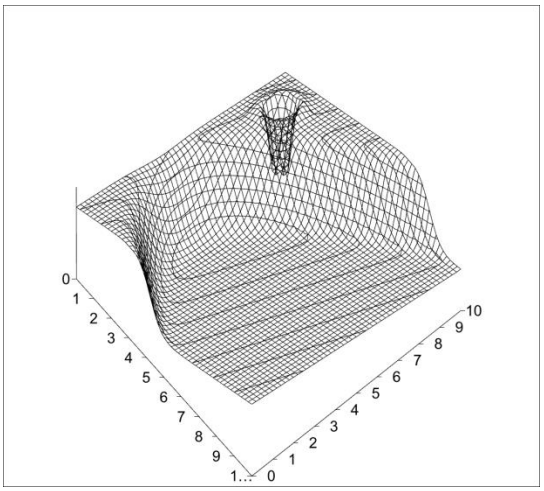


A feature of this application that creates difficulty is that the bottom of the print is a plane, and optimization algorithms that use a quadratic model (or second derivatives) cannot cope with the zero values of the second derivative.

The VBA code is:

```
If x1 < 0 Or x2 < 0 Or x1 > 10 Or x2 > 10 Then
    constraint = "FAIL"
    Exit Function
End If
x1line = 1 + 0.2 * (x2 - 4) ^ 2
deviation = (x1line - x1)
penalty = 5 * (1 / (1 + Exp(-3 * deviation)))   'logistic functionality
f_of_x = 0.5 * x1 - 0.2 * x2 + penalty + add_noise
f_of_x = 10 * (f_of_x - 0.3) / 6
```

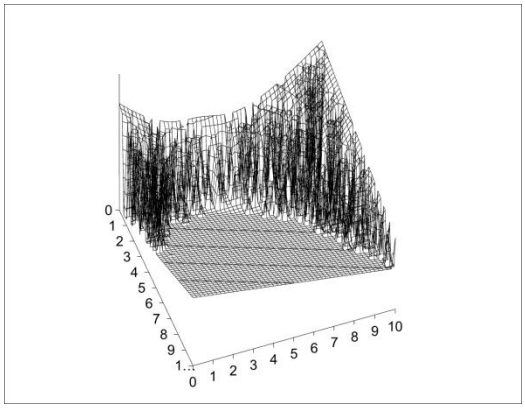
Boot Print with Pinhole (#22) – This is the same as Boot Print in the Snow, but the global minimum is entered with a small region on the level snow. Perhaps an acorn fell from a high tree and drilled a hole in the snow. The difficulty is that there is a small probability of starting in the region that would attract the solution to the true global. Nearly everywhere, the trial solution will be attracted to the toe part of the boot print.



The VBA Code is:

```
If x1 < 0 Or x2 < 0 Or x1 > 10 Or x2 > 10 Then
      constraint = "FAIL"
      Exit Function
End If
x1line = 1 + 0.2 * (x2 - 4) ^ 2
deviation = (x1line - x1)
penalty = 5 * (1 / (1 + Exp(-3 * deviation)))   'logistic functionality
f_of_x = 0.5 * x1 - 0.2 * x2 + penalty + add_noise
x1mc2 = (x1 - 1.5) ^ 2
x2mc2 = (x2 - 8.5) ^ 2
factor = 1 + (5 * (x1mc2 + x2mc2) - 2) * Exp(-4 * (x1mc2 + x2mc2))
f_of_x = factor * f_of_x + add_noise
f_of_x = 10 * (f_of_x - 0.3) / 6
```

Stochastic Boot Print (#20) – This represents the same water reservoir design problem as Boot Print in the Snow; however, the surface is stochastic. The OF value depends on a probability of the flood or draught event. Because of this each realization of the contour will yield a slightly different appearance. One realization of the 3-D view is shown. Note that starting in the middle of the DV space on the planar portion, a down-hill optimizer will progressively move toward the far side of the illustration where the spikes are. In that region of a small reservoir kept too full or too empty, there is a probability of encountering a costly flood or draught event that the reservoir cannot mitigate. Moving in the down-hill direction the optimizer may, or may not, encounter a costly event. If it does not, it continues to place trial solutions into a region with a high probability of a disastrous event, and continues into the bad region as the vagaries of probability generate fortuitous appearing OF values.

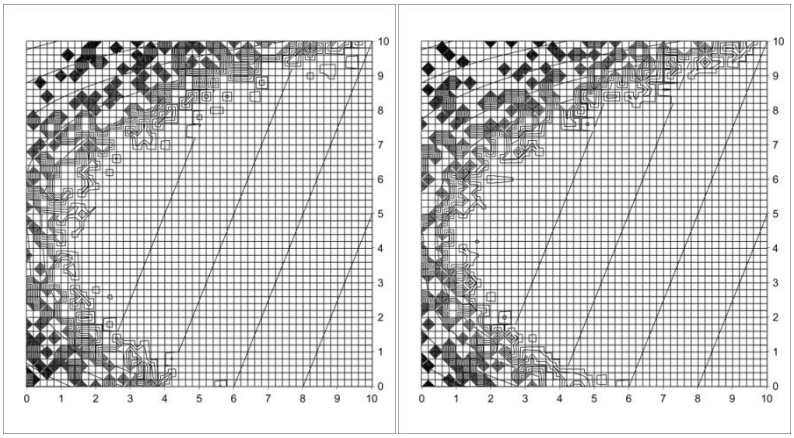


The VBA code is:

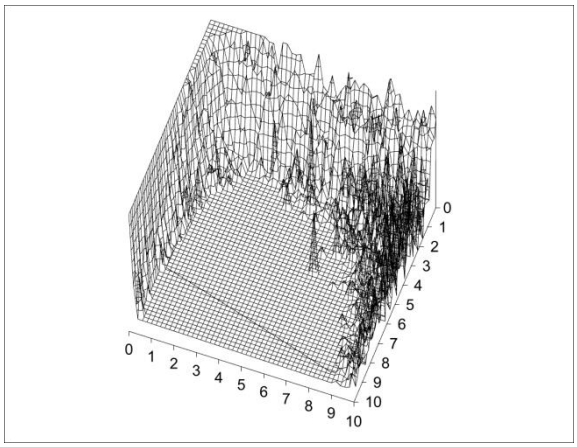
```
If x1 < 0 Or x2 < 0 Or x1 > 10 Or x2 > 10 Then
      constraint = "FAIL"
      Exit Function
End If
x1line = 2 + 0.2 * (x2 - 4) ^ 2
penalty = 0
```

```
deviation = (x1line - x1)
probability = 1 / (1 + Exp(-1 * deviation)) 'logistic functionality
If probability > Rnd() Then penalty = 5 * probability
f_of_x = 0.5 * x1 - 0.2 * x2 + penalty + add_noise
f_of_x = 10 * (f_of_x + 1.25) / 7.25
```

Two realizations of the contour are shown here and reveal the stochastic nature of the surface, the non-repeatability of the OF value. Now, in addition to the difficulty of the planar midsection, the optimizer also faces a stochastic surface that could lead to a fortuitous minimum in a high risk section of too small a dam (x-axis) or keeping the reservoir too full or empty (y-axis).



Reservoir (#18) – This is the basis for Stochastic Boot Print, but it is a computationally time-consuming Monte Carlo simulation of a water reservoir. Reservoir capacity and nominal level are the decision variables.



The larger the reservoir, the greater the initial cost. Cost is the objective function. So, superficially, build a small dam and have a small reservoir to reduce cost. However, if the reservoir is too small, and/or it is maintained nearly full, it does not have enough capacity to absorb an up-stream flood due to exceptionally heavy rainfall, and it will transmit the flood down-stream. Down-stream flooding incurs a cost of damaged property. But, the chance of a flood, and the magnitude of the flood depend on the up-stream rainfall. So, the simulator models a day-to-day status with a log-normal rainfall distribution for a time-period of 20 years. (You can change the simulated time, or rain fall distribution.)

On the other side of flooding conditions are drought conditions. If the reservoir is too small, and/or maintained with little reserve of water, an upstream drought will require stopping the water release, which stops down-stream river flow. Down-stream dwellers, recreationists, or water users will not like this. There is also a cost related to zero down-stream flow.

There is a fixed cost of the structure, and a probabilistic or stochastic cost of extreme flood or draught events.

Since one 20-year simulation will not reveal the confluence of "100-year events", the simulator runs 50 reservoirs for 20 years each – 50 realizations of the 20-year period. You can change the number of realizations.

The function is set-up to return either the maximum cost for the 50 realizations, or the estimated 99% probable upper limit on the cost. You could choose another performance indicator.

An excessively large reservoir kept half full will have ample reserve (to keep water flowing in a drought) and open capacity (to absorb excess rainfall and prevent down-stream flooding), but it will cost a lot. A smaller reservoir will have less cost. But, too small a reservoir will not prevent problems with drought or flood. So, there is an in between optimum size.

If the nominal volume is near the full mark, then the reservoir will not be able to absorb floods, but it will have plenty of capacity for a drought. If the nominal level is too low, it will be able to prevent a flood, but not keep water flowing for a drought event. So, there is also an in between setpoint capacity that is best. The optimum setpoint for the level might not be at 50%. It depends on whether the vagaries of rainfall make floods a bigger event than droughts.

For any given sized reservoir, the fuller it is kept the greater is the fresh water reserve and recreational area. So, other benefits are added as a negative penalty to the cost.

There is a fixed cost of the structure, a probabilistic or stochastic cost of extreme events, and a negative penalty for reserve and recreation benefits.

The figure is rotated to provide a good view of the surface. The optimum is in the upper left of the figure. The lower axis is x2, the water level nominal setpoint for the reservoir. At zero the reservoir is empty, at 10 it is completely full. The nearly vertical axis on the right is the reservoir size, zero is a nonexistent reservoir, 10 is large.

Similar to Stochastic Boot Print, this function presents optimizer difficulties of the planar midsection and a stochastic surface that could lead to a fortuitous minimum in a high risk section of too small a dam (x-axis) or keeping the reservoir too full or empty (y-axis). It is a more realistic Monte Carlo simulation than Stochastic Boot Print, but takes longer to compute, and provides the same issues for an optimizer as Stochastic Boot Print.

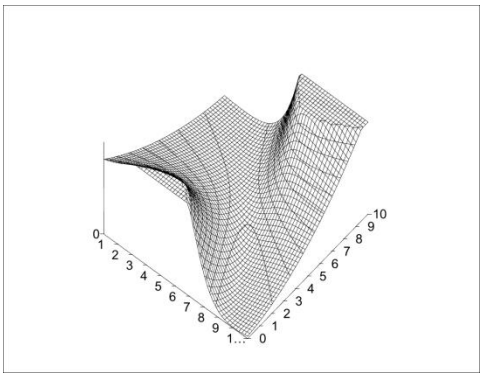
The VBA code is:

```
  twoPi = 2 * 3.1415926535
  inchavg = 0.4              'average inches of rainfall/event
  prain = 0.25               'daily probability of rain
  sy = Log(5 / inchavg) / 1.96   'sigma for log-normal distribution
  Qinavg = prain * inchavg * 2.54 * 10 ^ 6   'average water volume collected/day
  Useravg = 0.4 * Qinavg             'average daily water demand by users
  Qoutavg = Qinavg - Useravg         'average excess water/day released downstream
  Vc = (100 + x1 * 100) * 0.25 * inchavg * 2.54 * 10 ^ 6  'reservoir capacity, from scaled DV x1
  If Vc < 0 Then Vc = 0
  Vset = (0.3 + 0.65 * x2 / 10) * Vc          'reservoir setpoint from scaled DV x2
  Vmin = 0.3 * Vc                  'Minimum residual capacity in reservoir, a constraint
  inchflood = 12                   'rain fall amount at one time that causes a flood downstream
```

```
  Qflood = inchflood * 2.54 * 10 ^ 6      'volume of water/day associated with the flood rain
  v = Vset                          'initialize simulation with water at the setpoiny
  TotalDays = 1 * 365               'simulation period in days
  NRealizations = 10                'number of realizations simulated
  MaxCost = 0                       'initialize total cost for any realization
  For Realization = 1 To NRealizations
    Cost(Realization) = 0.0001 * Vc ^ 0.6       'cost of initial reservoir
    For Daynum = 1 To TotalDays
      If Rnd() > 0.25 Then                  'does it rain?
        inches = 0                      'if no
      Else
        inches = inchavg * Exp(sy * Sqr(-2 * Log(Rnd())) * Sin(twoPi * Rnd()))
        If inches > 25 Then inches = 25      'log-normal occasionally returns some excessive numbers. maybe rainfall is not log-normally distributed
      End If
      Qin = inches * 2.54 * 10 ^ 6          'incoming water volume due to rain - level times area
      If v <= Vmin Then Qout = 0            'control logic for water release
      If Vmin < v And v <= Vset Then Qout = Qoutavg * (v - Vmin) / (Vset - Vmin)
      If Vset < v And v <= Vc Then Qout = Qoutavg + (v - Vset) * (Qflood - Qoutavg) / (Vc - Vset)
      vnew = v + Qin - Qout - Useravg       'reservoir volume after a day if Qout as calculated
      If vnew > Vc Then                     'override if reservoir volume would exceed capacity
        Qout = Qin - Useravg - (Vc - v)
        vnew = Vc
      End If
      If vnew < Vmin Then                   'override if reservoir volume would fall lower than Vmin
        Qout = v - Vmin + Qin - Useravg
        If Qout < 0 Then Qout = 0
        vnew = v + Qin - Qout - Useravg
      End If
      v = vnew
      If Qout > Qflood Then                 'cost accumulation if a flood event
        discount = (1 + 0.03) ^ Int(Daynum / 365)   'discount factor
        Cost(Realization) = Cost(Realization) + (0.1 * ((Qout - Qflood) / 10 ^ 6) ^ 2) / discount
      End If
      If Qout = 0 Then                      'cost accumulation if a drought event
        discount = (1 + 0.03) ^ Int(Daynum / 365)
        Cost(Realization) = Cost(Realization) + 1 / discount
      End If
    Next Daynum
    If Cost(Realization) > MaxCost Then MaxCost = Cost(Realization)
  Next Realization
  costsum = 0           'determine average cost per realizaton
  For Realization = 1 To NRealizations
    costsum = costsum + Cost(Realization)
  Next Realization
  AvgCost = costsum / NRealizations
  cost2sum = 0          'determine variance of realization-to-relaization cost
  For Realization = 1 To NRealizations
```

```
    cost2sum = cost2sum + (Cost(Realization) - AvgCost) ^ 2
  Next Realization
  SigmaCost = Sqr(cost2sum / (NRealizations - 1))
  f_of_x = AvgCost + 3 * SigmaCost       'Primary OF is 3-sigma, 99.73 probable upper limit
  f_of_x = f_of_x - 0.2 * x2             'Secondary OF adds a benefit (negative penalty) for high setpoint level
  f_of_x = 10 * (Sqr(f_of_x) - 2) / (15 - 4) + add_noise          'scale factor for display convenience
'   f_of_x = MaxCost - 0.2 * x2 - 4      'OF based on max cost for the several realizations
  If Vset < Vmin Or Vset > 0.95 * Vc Or Vc <= 0 Then  'hard constraint
    constraint = "FAIL"
  End If
```

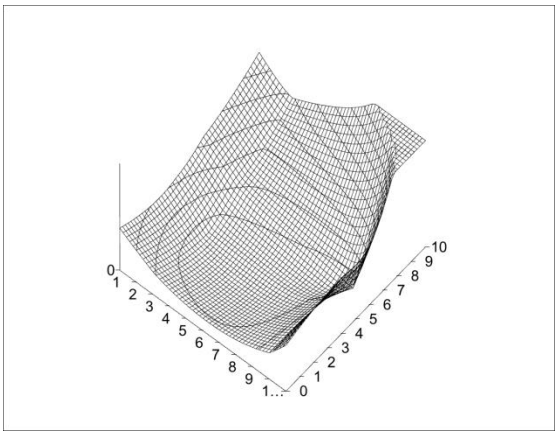
Chong Vu's Normal Regression (#11) – Chong Vu was a student exploring various regression objective functions as part of the Optimization Applications course, and created this test problem for a best linear relation to fit 5 data points representing contrived noisy data. The points are (0,1), (0,2), (1,3), (1,1), and (2,2). X1 and x2 are scaled to represent the slope and intercept of the linear model. The OF value is computed as the sum of squared normal distances from the line to the data (as opposed to the traditional vertical deviation least squares that assumes variability in the y-measurement only).



The minimum is at about x1=8, x2=2 in the near valley. The function is relatively well behaved, and even though it represents a sum of squared deviations, it is not a quadratic shape. Further, trial solutions in the far portion of the valley send the solution toward infinity (x1=-∞, x2=+∞).

The VBA Code is:

```
  m11 = 0.5 * x1 - 3       'coefficients adjusted to fit better on x1, x2 display scale
  b11 = 0.3 * x2 + 0.5
  Sum = 0
  Sum = Sum + (1 - m11 * 0 - b11) ^ 2    'first of 5 pairs of x,y data y=1, x=0
  Sum = Sum + (2 - m11 * 0 - b11) ^ 2
  Sum = Sum + (3 - m11 * 1 - b11) ^ 2
  Sum = Sum + (1 - m11 * 1 - b11) ^ 2
  Sum = Sum + (2 - m11 * 2 - b11) ^ 2
  f_of_x = Sum / (1 + m11 ^ 2) - 2        'sum of vertical distance squared converted to normal d^2
'   f_of_x = 0.6 * (f_of_x - 1 + 0.02 * (x1 - x2 + 3) ^ 2)  'OF adjusted for display appearance and to keep solution in bounds
  If x1 < 0 Or x2 > 10 Then constraint = "FAIL"          'there is an off-graph attractor seems to be at -infinity, + infinity
```



Windblown (#61) – A person wants to travel from Point A to Point B, and chooses two linear paths from A to Point C then C to B so that the cumulative impact of the wind is minimized. Perhaps he is not wearing his motorcycle helmet, and doesn't want his hair to get messed up. The DVs are the x1 and x2 coordinates for point C. The wind blows in a constant (not stochastic) manner, but the wind velocity and direction change with location. The travel velocity is constant. If point C is out of the high windy area, but far away, the travel time is high and the low wind experience persists for a long time. If point C is on the line between A and B, representing the shortest distance path, and lowest time path, it takes the traveler into the high wind area, and even though the time is minimized, the cumulative wind damage is high. The impact is due to the square of the difference of the wind and travel velocity. Moving at 25 mph in the same direction as a 25 mph wind is blowing is like being in calm air. But, traveling in the opposite direction is like standing in a 50 mph wind. The objective is to minimize the cumulative impact, the integral of the squared velocity difference along the A-C-B path.

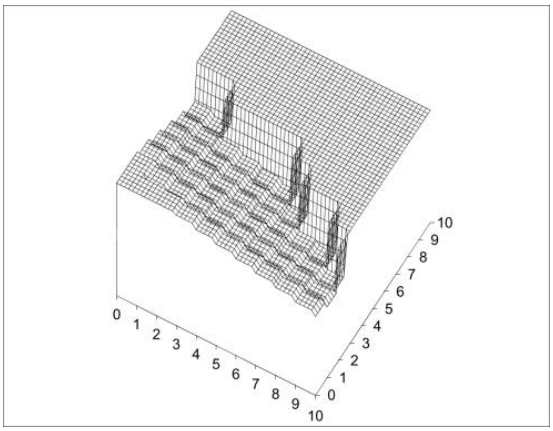
The function provides some discontinuities as evidenced by the kinks in the contours. And, there are two minima, the global is to the front-left of the lower contour, and the secondary is to the back right. Both minima are in relatively flat spots.

Suresh Kumar Jayaraman helped me explore this simulation of a path integral. The VBA code is:

```
xc = x1     'Optimizer chooses point C
yc = x2
xa = 2      'User defines points A and B
ya = 4
xb = 9
yb = 6
N = 200     'Discretization number of intervals
velocity = 1.5        'Velocity of traveller
wind_coefficient = 0.1          'Coefficient for velocity of wind
                        'from A to C
xi = xa     'xi and yi are locations along the path
yi = ya
D1 = Sqr((xa - xc) ^ 2 + (ya - yc) ^ 2)     'total path distance
dD1 = D1 / N                        'path incremental length
time_interval = dD1 / velocity          'travel time along path increment
dxi1 = (xc - xa) / N                'incremental x increments
dyi1 = (yc - ya) / N                'incremental y increments
If xc = xa Then
   vx = 0                           'traveler x velocity
Else
   vx = velocity * (1 + ((yc - ya) / (xc - xa)) ^ 2) ^ -0.5
End If
If yc = ya Then
   vy = 0                           'traveler y velocity
```

```
Else
   vy = velocity * (((xc - xa) / (yc - ya)) ^ 2 + 1) ^ -0.5
End If
im1 = 0                          'integral of impact on path 1
For Path_Step = 1 To N
   If D1 = 0 Then Exit For
   xi = xi + dxi1
   yi = yi + dyi1
   wx = (-wind_coefficient * xi ^ 2 * yi) / Sqr((xi + 0.1) ^ 2 + (yi + 0.1) ^ 2)
   wy = (wind_coefficient * xi * yi ^ 2) / Sqr((xi + 0.1) ^ 2 + (yi + 0.1) ^ 2)
   lim = Sqr((vx - wx) ^ 2 + (vy - wy) ^ 2)
   im1 = im1 + lim
Next Path_Step
im1 = im1 * time_interval              'total impact scaled by time
                        'from C to B
xi = xc
yi = yc
D2 = Sqr((xc - xb) ^ 2 + (yc - yb) ^ 2)
dD2 = D2 / N
time_interval = dD2 / velocity
dxi2 = (xb - xc) / N
dyi2 = (yb - yc) / N
If xc = xb Then
   vx = 0
Else
   vx = velocity * (1 + ((yb - yc) / (xb - xc)) ^ 2) ^ -0.5
End If
If yc = yb Then
   vy = 0
Else
   vy = velocity * (((xb - xc) / (yb - yc)) ^ 2 + 1) ^ -0.5
End If
im2 = 0
For Path_Step = 1 To N
   If D2 = 0 Then Exit For
   xi = xi + dxi2
   yi = yi + dyi2
   wx = (-wind_coefficient * xi ^ 2 * yi) / Sqr((xi + 0.1) ^ 2 + (yi + 0.1) ^ 2)
   wy = (wind_coefficient * xi * yi ^ 2) / Sqr((xi + 0.1) ^ 2 + (yi + 0.1) ^ 2)
   lim = Sqr((vx - wx) ^ 2 + (vy - wy) ^ 2)
   im2 = im2 + lim
Next Path_Step
im2 = im2 * time_interval
f_of_x = im1 + im2
f_of_x = 10 * (f_of_x - 18) / (35 - 18)
f_of_x = f_of_x + add_noise
```

Integer Problem (#33) – This simple example represents a classic manufacturing application. Minimize a function (perhaps maximize profit) of DVs x1 and x2 (perhaps the number of items of products A and B to make), subject to constraints (perhaps on capacity), and requiring x1 and x2 to be integers (you can only sell whole units).



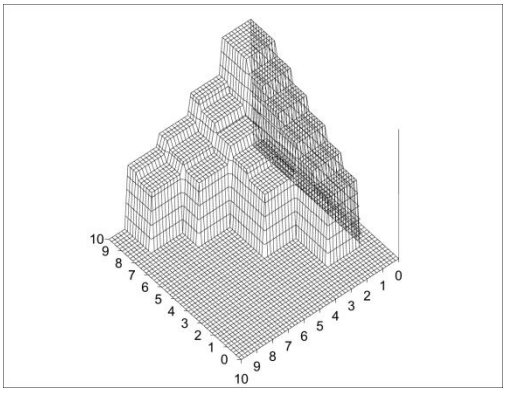
There are many similar examples in textbooks.

The attributes of such applications are the level discontinuities (cliffs) as the integer value changes, and the flat spots over the range of DV values that generate the same integer value. The surface is nonanalytic – derivatives are either zero or infinity. This illustration illustrates the constraint regions with a high OF value.

The VBA Code is:

```
x11 = Int(x1)
x22 = Int(x2)
f_of_x = 11 - (4 * x11 + 7 * x22) / 10
If 3 * x11 + 4 * x22 > 36 Then constraint = "FAIL"
If x11 + 8 * x22 > 49 Then constraint = "FAIL"
```

Reliability (#56) – This application represents a designer's choice of the number and size of parallel items for system success. Consider a bank of exhaust fans needed to keep building air refreshed, as I had to when I worked in industry. The fans are operating in parallel. If one fails, air quality deteriorates. If there are 3 operating fans and one spare; when one fails, the spare can be placed online. This increases reliability of the operation, but increases the cost of the fan assembly by 4/3. Even so, reliability is not perfect. There is a chance that two fans will fail, or three. Perhaps have three spares. Now, the cost is 6/3 of the original fan station. In either case, the cost needs to be balanced by the probability of the fan bank not handling the load. Alternately, it could be balanced by risk (the financial penalty of an event, times the probability that an event will happen).



A clever cost reduction option is to use smaller capacity items, but more of them. For example, if there are 4 operating fans, each only has to have ¾ of the capacity of the original three. Using the common 6/10^{ths} power law for device cost, the smaller fans each cost $(3/4)^{0.6}$ of the original three, but there are 4 of them. So, the cost of the zero-spare situation with 4 smaller fans is higher than the cost of the 3 larger fans. The ratio is $4^* (3/4)^{0.6} /3 = 1.12$. However, if three spares are adequate, the cost of 7 smaller fans is lower than the cost of 6 larger fans. The ratio is $7^* (3/4)^{0.6} /6 = 0.98$.

The optimization objective is to determine the number of operating units and the number of spare units to minimize cost with a constraint that the system reliability must be greater than 99.99%.

The realizable values of the DVs must be integers. This creates flat surfaces with cliff discontinuities (derivatives are either zero or infinity), and the constraint creates infeasible DV sets.

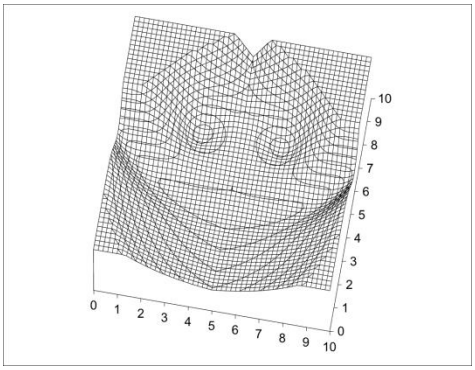
The VBA Code is:

```
  N = 3 + Int(8 * x1 / 10 + 0.5) 'total number of components
  M = Int(5 * x2 / 10 + 0.5) 'number of operating components needed to meet capacity
  p = 0.2          'probability of any one component failing
  q = 1 - p        'probability of any one component working
  If M > N Then
    f_of_x = 0
    constraint = "FAIL"
    Exit Function
  End If
  If M > 0 Then
    f_of_x = N * (1 / M) ^ 0.6
  Else
    f_of_x = 0
    constraint = "FAIL"
    Exit Function
  End If
  P_System_Success = 0
  For im = 0 To N - M
    P_System_Success = P_System_Success + (factorial(N) / (factorial(im) * factorial(N - im))) * (p ^ im) *
(q ^ (N - im))
  Next im
  If P_System_Success < 0.999 Then
    f_of_x = 0
    constraint = "FAIL"
    Exit Function
  End If
```

The factorial function is:

```
  factorial = 1
  If NUM = 1 Or NUM = 0 Then Exit Function
  For iNUM = 2 To NUM
    factorial = factorial * iNUM
  Next iNUM
```

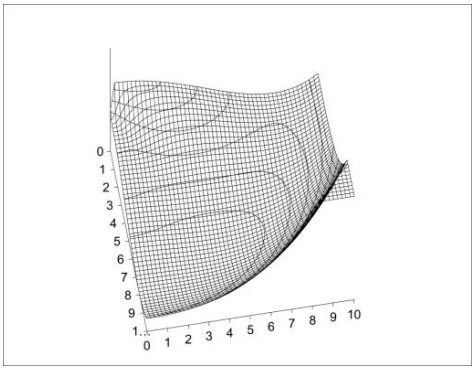
Frog (#2) – I generated this function as a final project for a freshman-level computer programming class when I was a PhD candidate at NCSU. It included nested loops and conditionals to assign text symbols to array variables, then printing the array. Students would know when they have the right answer! The eyes represent equal global optima. There are also three minima in the mouth. To add difficulty for an optimizer, there is an oval constraint, a forbidden area, surrounding the eye in the upper left. Down-hill type optimizers get stuck on the constraint northwest of the eye. The face is also relatively flat tricking some convergence criteria to stop early.



The VBA Code is:

```
f_of_x = 2 + 1 * (((x1 - 5) ^ 2) + ((x2 - 5) ^ 2)) * (0.5 - Exp((-((x1 - 3.5) ^ 2)) - _
    ((x2 - 7) ^ 2))) * (0.5 - Exp((-((x1 - 6.5) ^ 2)) - _
    ((x2 - 7) ^ 2))) * (0.5 + (Abs(Sqr(((x1 - 5) ^ 2) / ((x2 - 11) ^ 2)))) - _
    (Exp(-(Sqr(((x1 - 5) ^ 2) + ((x2 - 11) ^ 2)) - 7) ^ 2)))
If (x1 - 3.5) ^ 2 + (x2 - 7) ^ 4 <= 2 Then
    constraint = "FAIL"                          'Hard constraint approach
Else
    constraint = "OK"
End If
```

Hot and Cold Mixing (#36) – This function represents the control action required to meet steady-state mixed temperature and flow rate targets of 70 C and 20 kg/min from the current conditions of 35C and 8 kg/min. The DVs are the signals to the hot and cold valves.



Hot and cold fluid are mixed in line, and the objective is to determine the hot and cold flow control valve positions, o_1 and o_2, to produce the desired mixed temperature and total flow rate. The valves have a parabolic inherent characteristic, and identical flow rate vs. valve position response. The control algorithm is the Generic Model Control (GMC) law with a steady-state model, and the desire to target for 20% beyond the biased setpoint, Kc=1.2. There is uncertainty on the model parameters of the supply temperatures, measured flow rate and temperature, and valve Cv. The controller objective is to determine o_1 and o_2 values that minimize the equal-concern weighted deviations from target at steady-state.

$$J = \frac{\left[K_c\left(T_{sp}-T_p\right)+T_p-\frac{T_h {o_1}^2+T_c {o_2}^2}{{o_1}^2+{o_2}^2}\right]^2}{{E_T}^2} + \frac{\left[K_c\left(F_{sp}-F_p\right)+F_p-C_v\left({o_1}^2+{o_2}^2\right)\right]^2}{{E_F}^2}$$

The first term in the OF relates to the temperature deviation, and the second term the flow rate deviation. The deviations are weighted by the equal-concern factors, E_T and E_F. The numerator of each term starts

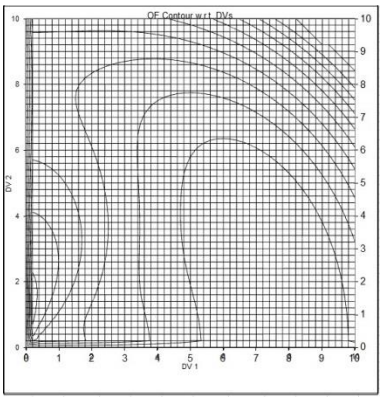
with the calculated target value (beyond the setpoint), and subtracts from it the modeled value. The OF is the equal-concern-weighted, sum of squared deviations.

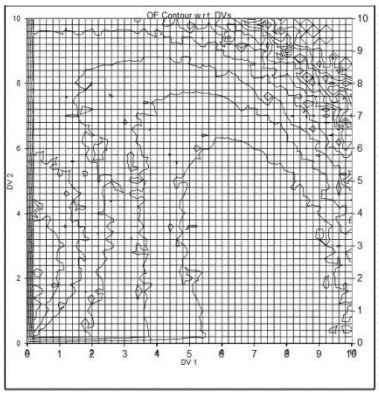
The VBA code is:

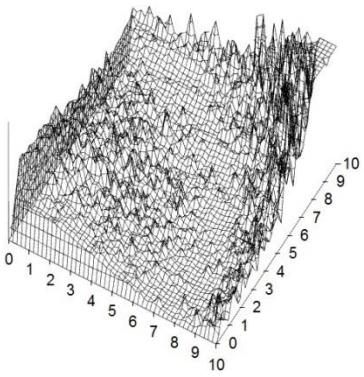
```
If x1 <= 0 Or x1 > 10 Or x2 <= 0 Or x2 > 10 Then
      constraint = "FAIL"
      Exit Function
   End If
   o1 = 10 * x1       'hot valve position, %
   o2 = 10 * x2       'cold valve position, %
   SetpointT = 70                'Celsius
   add_noise = Worksheets("Main").Cells(8, 12) * Sqr(-2 * Log(Rnd())) * Sin(2 * 3.14159 * Rnd())
   FromT = 35 * (1 + add_noise)       'Celsius
   SetpointF = 20                'm^3/min
   add_noise = Worksheets("Main").Cells(8, 12) * Sqr(-2 * Log(Rnd())) * Sin(2 * 3.14159 * Rnd())
   FromF = 8 * (1 + add_noise)        'm^3/min
   add_noise = Worksheets("Main").Cells(8, 12) * Sqr(-2 * Log(Rnd())) * Sin(2 * 3.14159 * Rnd())
   HotTin = 80 * (1 + add_noise)      'Celsius
   add_noise = Worksheets("Main").Cells(8, 12) * Sqr(-2 * Log(Rnd())) * Sin(2 * 3.14159 * Rnd())
   ColdTin = 20 * (1 + add_noise)     'Celsius
   add_noise = Worksheets("Main").Cells(8, 12) * Sqr(-2 * Log(Rnd())) * Sin(2 * 3.14159 * Rnd())
   ValveCv = 0.0036 * (1 + add_noise)  'm^3/min/%^2
   EC4T = 0.15                   'Celsius^(-2)
   EC4F = 1                     '(m^3/min)^(-2)
   f_of_x = EC4T * (1.2 * (SetpointT - FromT) + FromT - (HotTin * o1 ^ 2 + ColdTin * o2 ^ 2) / (o1 ^ 2 + o2 ^
2)) ^ 2 + EC4F * (1.2 * (SetpointF - FromF) + FromF - ValveCv * (o1 ^ 2 + o2 ^ 2)) ^ 2
   f_of_x = f_of_x / 150
```

This is a simple function, but provides substantial misdirection to a steepest descent optimizer that starts in the far side. It has steep walls, but a low slope at the proximity of the minimum. Some optimizers starting in the proximity of the optimum do not make large enough DV changes, and convergence criteria can stop them where they start.

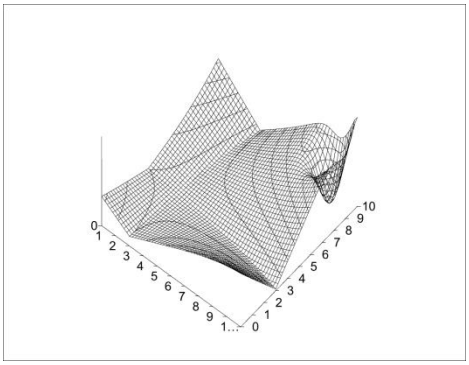
With no uncertainty on model values, the contour of the 2-D search or o1 and o2 appears as the left figure below, which is interesting enough as a test case for nonlinear optimization. However, with a 5% nominal uncertainty, the center figure, the contours are obviously irregular, and the 3-D plot of OF vs. DVs, the right hand figure, reveals the irregular surface.







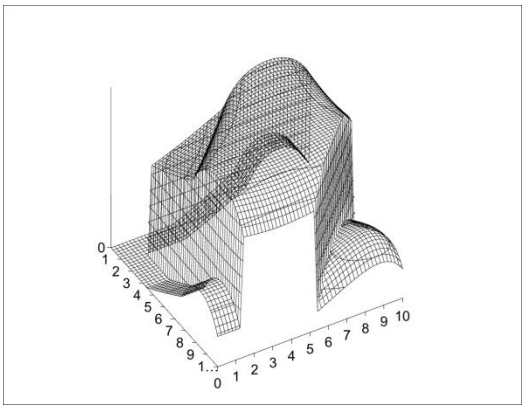
Curved Sharp Valleys (#8) – This is a contrivance to provide a simple function with a slope discontinuity at the global minimum (in the valley near the lower right of the figure, but in the interior, at about the point x1=8, x2=3). At the minimum the slope of the valley floor is low compared to the side walls. This means that from any point in the bottom of the valley there is only a small directional angle to move to a lower spot. Nearly all directions point up hill. Also the valley is curved, so that once the right direction is found, it is not the right direction for the next move. Most optimizers will "think" they have converged when they are in the steep valley, and multiple runs will lead to multiple ending points that trace the valley bottom. The surface has another minimum in a valley in the upper right, and a well behaved local minimum up on the hill in the far right. Both the Parameter Correlation and the ARMA Regression function have a similar feature. This exaggerates it, and has a very simple formulation.



The VBA Code is:

```
f_of_x = 0.015 * (((x1 - 8) ^ 2 + (x2 - 6) ^ 2) + _
    15 * Abs((x1 - 2 - 0.001 * x2 ^ 3) * (x2 - 4 + 0.001 * x1 ^ 3)) - _
    500 * Exp(-((x1 - 9) ^ 2 + (x2 - 9) ^ 2)))
```

Parallel Pumps (#41) – This represents a redundancy design application from a 2011 assignment worded as "A company has three identical centrifugal pumps in parallel in a single process stream. The pumps run at a constant impeller speed. They wish to have a method that chooses how many pumps should be operating, and what flow rate should go through each operating pump to minimize energy consumption for a given total flow rate. The inlet and outlet pressures on the overall system of pumps remain a constant, but not on each individual pump. The individual flow rates out of each pump are controlled by a flow control valve.



"The pump characteristic curves (differential pressure vs. flow rate) are described as:

$$\Delta P_i = 1.22\,\Delta P_{nominal}(1 - 0.017e^{2.37F_i/F_{nominal}})$$

That means that at high flow rates, the outlet pressure will be low. The choice of the pump flow rate must create a pressure that is greater than the down-stream line pressure.

"The efficiency of each pump is roughly a quadratic relation to flow rate.

$$\epsilon_i = .7r_i(2 - r_i),\ \ r_i = {}^{F_i}/_{F_{nominal}}$$

The nominal flow rate rating is at the peak efficiency, but the pumps can operate at a flow rate that is 50% above nominal, $F_{maximum} = 1.5\,F_{nominal}$. The energy consumption for any particular pump is the flow rate times the pressure drop divided by efficiency. And the total power is the sum for all operating pumps.

$$\dot{E} = \sum E_\iota = \sum \frac{\Delta P_i F_i}{\epsilon} = \frac{1.22\,\Delta P_{nominal}\,F_n{}^2}{0.7} \sum \frac{(1 - 0.017e^{2.37F_i/F_{nominal}})}{(2F_n - F_i)}$$

"We prefer to not run a pump if its flow rate is less than 20% of nominal, because of the very low energy efficiency. We also wish to not use all three pumps unless absolutely needed - we have a fairly strong desire to keep a spare pump off-line for maintenance so that we always have one in peak condition. Although, it might be best for energy efficiency when the target total flow rate is $3*F_{nominal}$ to run all three pumps at $F_{nominal}$, the desire to keep a spare would prefer that we operate two pumps at 1.5 times $F_{nominal}$."

This OF reveals surface discontinuities, cliffs, due to the switching of pumps, and multiple optima.
The VBA Code provides an option for either hard or soft constraints on the issue of running pumps when they would be less than 20% of nominal. The DVs, x1 and x2 represent the fraction of maximum of any two pumps. The third pump makes up the balance.

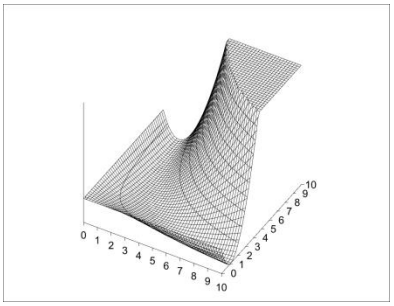
The code is:

```
Qnominal = 6
Qmaximum = 1.5 * Qnominal
Qminimum = 0.2 * Qnominal
If x1 < 0 Or x2 < 0 Or x1 > 10 Or x2 > 10 Then constraint = "FAIL"
Qtotal = 15
o1 = x1 * Qmaximum / 10        'Transfer to permit an override if Q<min
o2 = x2 * Qmaximum / 10
penalty = 0
ConstraintType = "Hard"       '  "Soft"   ' Preference on running pumps less than Qminimum
If ConstraintType = "Soft" And o1 < 0.1 * Qminimum Then o1 = 0  'Absolutely off if a trivial flow
rate
If ConstraintType = "Hard" And o1 < Qminimum Then o1 = 0
If ConstraintType = "Soft" And o1 < Qminimum Then penalty = penalty + 2 * (o1 - 0) ^ 2      'What
to use for equal concern factors?
If ConstraintType = "Soft" And o2 < 0.1 * Qminimum Then o2 = 0
If ConstraintType = "Hard" And o2 < Qminimum Then o2 = 0
If ConstraintType = "Soft" And o2 < Qminimum Then penalty = penalty + 2 * (o2 - 0) ^ 2
o3 = Qtotal - o1 - o2
```

```
If o3 < 0 Or o3 > Qmaximum Then constraint = "FAIL"
If o3 > Qmaximum Then o3 = Qmaximum
If ConstraintType = "Soft" And o3 < 0.1 * Qminimum Then o3 = 0
If ConstraintType = "Hard" And o3 < Qminimum Then o3 = 0
If ConstraintType = "Soft" And o3 < Qminimum Then penalty = penalty + 2 * (o3 - 0) ^ 2
If Abs(o1 + o2 + o3 - Qtotal) > 0.0001 * Qtotal Then constraint = "FAIL" 'hard constraint on production
If ConstraintType = "Hard" And 1.22 * (1 - 0.017 * Exp(2.37 * o1 / Qnominal)) < 0.7 Then constraint = "FAIL"  'Constraint on output pressure
If ConstraintType = "Hard" And 1.22 * (1 - 0.017 * Exp(2.37 * o2 / Qnominal)) < 0.7 Then constraint = "FAIL"
If ConstraintType = "Hard" And 1.22 * (1 - 0.017 * Exp(2.37 * o3 / Qnominal)) < 0.7 Then constraint = "FAIL"
If ConstraintType = "Soft" And 1.22 * (1 - 0.017 * Exp(2.37 * o1 / Qnominal)) < 0.7 Then penalty = penalty + 10 * (1.22 * (1 - 0.017 * Exp(2.37 * o1 / Qnominal)) - 0.7) ^ 2
If ConstraintType = "Soft" And 1.22 * (1 - 0.017 * Exp(2.37 * o2 / Qnominal)) < 0.7 Then penalty = penalty + 10 * (1.22 * (1 - 0.017 * Exp(2.37 * o2 / Qnominal)) - 0.7) ^ 2
If ConstraintType = "Soft" And 1.22 * (1 - 0.017 * Exp(2.37 * o3 / Qnominal)) < 0.7 Then penalty = penalty + 10 * (1.22 * (1 - 0.017 * Exp(2.37 * o3 / Qnominal)) - 0.7) ^ 2
f1 = 0
f2 = 0
f3 = 0
If o1 > 0 Then f1 = (1 - 0.017 * Exp(2.37 * o1 / Qnominal)) / (2 * Qnominal - o1)  'Calculate scaled energy rate
If o2 > 0 Then f2 = (1 - 0.017 * Exp(2.37 * o2 / Qnominal)) / (2 * Qnominal - o2)
If o3 > 0 Then f3 = (1 - 0.017 * Exp(2.37 * o3 / Qnominal)) / (2 * Qnominal - o3)
If o1 * o2 * o3 > 0 Then penalty = penalty + 1      'soft penalty if all three pumps are running
```

Underspecified (#43) – There are an infinite number of simple functions to create that are underspecified, that permit infinite solutions with the same OF value. These arise in optimization applications that are missing one (or more) of the impacts of a DV, when the OF is incomplete. In this one the code is:

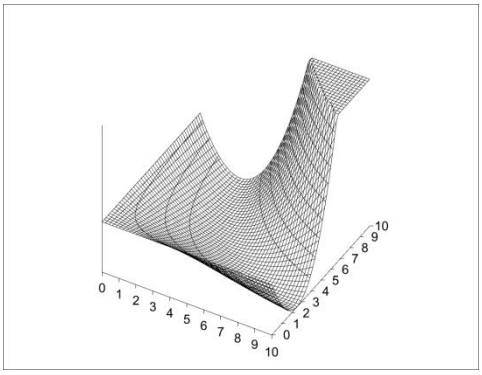
```
f_of_x = 2 * (x1 * x2 / 20 - Exp(-x1 * x2 / 20)) ^ 2
```



Since x1 and x2 always appear as a product, the function does not have independent influence by the DVs. Regardless of the value of x1, as long as the value of x2 makes the product equal 11.343, the function is at its optimum.

Parameter Correlation (#54) – This is similar to the underspecified case, but brought about by an extreme condition that makes what appears to be independent functionalities reduce to a joint influence. I encountered this in modeling viscoelastic material using a hyper-elastic model for a nonlinear spring. The model computes stress, σ, given strain, ϵ.

$$\sigma = A(e^{B\varepsilon} - 1)$$

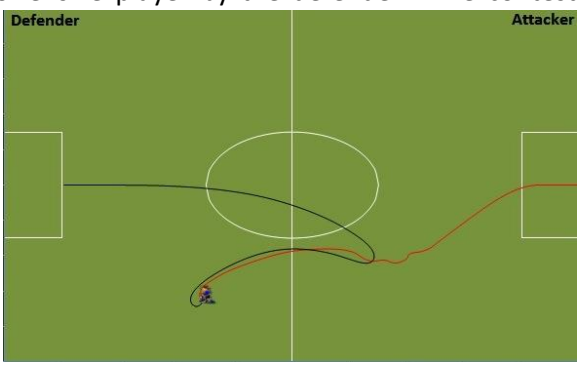


The DVs, A and B appear independent. But if ϵ is low, the product Bϵ is small, and in the limit, the exponential becomes $e^{B\varepsilon} \cong 1 + B\varepsilon$ resulting in $\sigma \cong AB\varepsilon$ in which the value of coefficients A and B are dependent. Run many trials and plot the values of A w.r.t. B, and there will be a strong correlation. The 3-D plot happens to be very similar to that of the underspecified example, but there is a true minimum in the valley.

The VBA Code is:

```
epsilon1 = 0.01        'no problem if epsilons = 0.1 and 0.2.  Then it doesn't degenerate to a product
epsilon2 = 0.02
‘ the data are generated by three true values of x1=5 and x2=5.
‘minimize the model deviations from data.
f_of_x = (x1 * (Exp(x2 * epsilon1) - 1) - 5 * (Exp(5 * epsilon1) - 1)) ^ 2 + (x1 * (Exp(x2 * epsilon2) - 1) - 5 * (Exp(5 * epsilon2) - 1)) ^ 2
f_of_x = 10 * f_of_x    'minimize normal equations deviation from zero
```

Robot Soccer (#35) – The Automation Society at OSU (the Student Section of the International Society of Automation at OSU) creates an automation contest each year. In 2011 it was to create the rules for a simulated soccer defender to intercept the offensive opponent dribbling the ball toward the goal. The players start in randomized locations on their side of the field, and the offensive player has randomized evasive rules. Both players are subject to limits on speed and momentum changes. The ball-carrying opponent moves slightly slower than the defensive player. The figure shows one realization of the path of the two players, and the final interception of the offensive player by the defender. The contest objective was to create motion rules (vertical and horizontal speed targets) for the defender to intercept the opponent. The OF was to maximize the fraction of successful stops in 100 games. If a perfect score, then the objective was to minimize the average playing time to intercept the opponent.

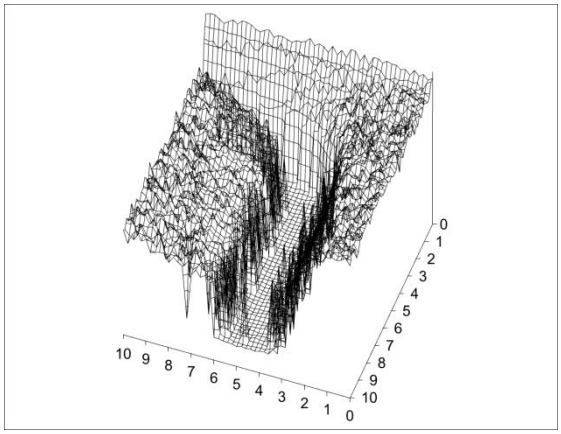


Much thanks to graduate student Solomon Gebreyohannes for creating the code and simulator.

Simple motion rules for the defender might be: Always run as fast as possible. And, run directly toward the offensive player. But in this case, with too much momentum, and evasive move by the offensive player may permit the ball-carrier to get to the goal when momentum carries the defender past.

The defender motion rules in the code below are: Extrapolate the opponent speed and direction to anticipate where it will be in "slead" seconds. Run toward that spot at a speed that gets you there in "stemper" reciprocal seconds. X1 and x2 are scaled variables for "slead" and "stemper". The optimizer objective is to determine x1 and x2 to maximize fraction of successful stops, and if 100%, then to minimize playtime. This is a stochastic problem. x1, the lead time is on the right-hand axis, and x2, the speed temper factor is on the near axis. The rotation is chosen to provide a best view of the surface. The successful region is seen as a curving, and relatively flat valley within steep walls.

The high surface is stochastic, the value of any point changes with each sampling; and optimizers that start there will wander around in confusion seeking a local fortuitous best.



However, the floor of the valley is also stochastic. It represents the game time for speed rules that are 100% successful, and replicate DV values do not produce identical OF values. Accordingly, optimizers based on surface models will be confounded.

Further, the motion rules limit the defender speed, so that regardless of combinations of desired motion rules the player cannot move any faster, and the floor of the valley is truly flat. It would be underspecified, except for the stochastic OF fluctuations.

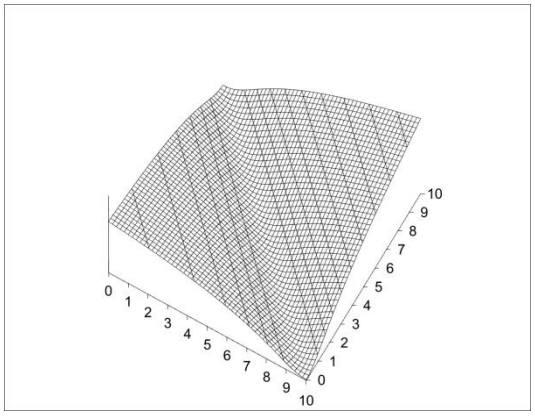
The VBA Code is:

```
  Dim GameTimeCounter As Integer
  If x1 < 0 Or x1 > 10 Or x2 < 0 Or x2 > 10 Then
    constraint = "FAIL"
    Exit Function
  End If
  slead = 3 * (x1 - 3) 'DV 1
  stemper = (x2 - 0) / 10 'DV 2
  NumGames = 50   '500
  GameTimeCounterMax = 10000
  Distance = 0
  SuccessCount = 0
  undercount = 0
  sdt = 0.03
  Lamdad = 0.2            'momentum constraints on speed rate of change
  CLamdad = 1 - Lamdad
  Lamdaa = 0.2
  CLamdaa = 1 - Lamdaa
  AAngle = 3.14159
  For GameNumber = 1 To NumGames
    Randomize
    'Player defender initialize
    xd = 1
    yd = 5
```

```
    vxd = 0
    vyd = 0
    'Player attacker initialize
    xa = 8 + 2 * Rnd()   'Int(Rnd() * 3)
    ya = 10 * Rnd()     'Int(Rnd() * 11)
    vxa = -2 * Rnd()
    vya = 2 * (Rnd() - 0.5)
    For GameTimeCounter = 1 To GameTimeCounterMax
      'Changing the attacker vx and vy speed
      If xa > 3 Then
        separation = Sqr((xa - xd) ^ 2 + (ya - yd) ^ 2)
        If separation < 3 And xa > xd Then
          LoAngle = 2 * 3.14159 * 45 / 360
          HiAngle = 2 * 3.14159 * 315 / 360
          If GameTimeCounter = 3 * Int(GameTimeCounter / 3) Then AAngle = LoAngle + Rnd() *
(HiAngle - LoAngle)
        Else
          LoAngle = 2 * 3.14159 * 120 / 360
          HiAngle = 2 * 3.14159 * 250 / 360
          If GameTimeCounter = 10 * Int(GameTimeCounter / 10) Then AAngle = LoAngle + Rnd() *
(HiAngle - LoAngle)
        End If
      Else
        LoAngle = 3.14159 + Atn((6.5 - ya) / (-xa))
        HiAngle = 3.14159 + Atn((3.5 - ya) / (-xa))
        If GameTimeCounter = 3 * Int(GameTimeCounter / 3) Then AAngle = LoAngle + Rnd() * (HiAngle
- LoAngle)
      End If
      ASpeed = 4 + Rnd() * (5 - 4)
      vxa = CLamdaa * vxa + Lamdaa * ASpeed * Cos(AAngle)
      vya = CLamdaa * vya + Lamdaa * ASpeed * Sin(AAngle)
      xa = xa + vxa * sdt       'Calculate current position of player 2 (Attacker)
      ya = ya + vya * sdt
      'Field constraints
      If ya > 10 Then ya = 10
      If ya < 0 Then ya = 0
      If xa > 10 Then xa = 10
      If xa < 0 Then xa = 0
     '  Defender rules - Anticipate where target will be at next interval and seek to get there
      xdtarget = xa + slead * sdt * vxa          'anticipated attacker location leadnumber of dts intot he
future
      ydtarget = ya + slead * sdt * vya
      Ux = stemper * (xdtarget - xd) / sdt   'speed to get defender there - tempered faster or slower
      Uy = stemper * (ydtarget - yd) / sdt
      Utotal = Sqr(Ux ^ 2 + Uy ^ 2)
      If Utotal > 5 Then
        Ux = Ux * 5 / Utotal              'constrained defender target speed, 5 is the max
        Uy = Uy * 5 / Utotal
```

```
        End If
        'Calculate new speed and position of the defender using calculated Ux and Uy
        vxd = CLamdad * vxd + Lamdad * Ux
        vyd = CLamdad * vyd + Lamdad * Uy
        xd = xd + vxd * sdt
        yd = yd + vyd * sdt
        'Field constraints
        If yd > 10 Then yd = 10
        If yd < 0 Then yd = 0
        If xd > 10 Then xd = 10
        If xd < 0 Then xd = 0
        If xa <= 0 And ya >= 3.5 And ya <= 6.5 Then Exit For    'Goal scored
        'Check if both players are in 0.2 radius region
        If Sqr((xd - xa) ^ 2 + (yd - ya) ^ 2) < 0.2 Then
          SuccessCount = SuccessCount + 1
          PlayTime = PlayTime + sdt * GameTimeCounter              'play time to intercept
          Distance = Distance + Sqr((xa - 0) ^ 2 + (ya - 5) ^ 2)      'stopped this distance from goal
          Exit For
        End If
      Next GameTimeCounter
    Next GameNumber
    AvgDistance = Distance / NumGames          'larger is better
    AvgPlayTime = PlayTime / NumGames          'smaller is better
    SuccessRatio = SuccessCount / NumGames
    If SuccessRatio = 1 Then                   'Only important if zero goals scored
      f_of_x = AvgPlayTime
    Else
      f_of_x = 10 - 5 * SuccessRatio   'if a goal is scored, then playtime and distance are unimportant
    End If
```

ARMA(2,1) Regression (#62) – This optimization seeks the coefficients in an equation to model a second-order process. The true process is $y_i = a*y_{i-1} + b* y_{i-2} + u_{i-1}$. Here y is the process response, u is the process independent variable, influence, and i is the time counter. The term u, the forcing function, is expressed as "push" in the VBA assignment statements. The model has the same functional form as the process. The optimizer objective is to determine model coefficient values that minimize the sum of squared differences between data and model. A good optimizer will find the right values of a=0.3 and b=0.5.

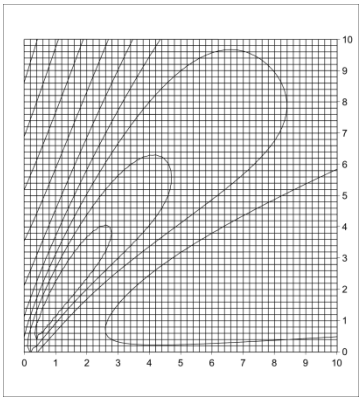


The difficulty of this problem is that the true solution is in the middle of a long valley of steep walls and very gentle longitudinal slope. Like the Steep Valley and Parameter Correlation functions, when the trial solution is in the bottom of the valley, optimizers have difficulty in moving it along the bottom to the optimum.

The VBA Code is:

```
  x11 = 0.3 + (x1 - 5) / 100
  x22 = 0.5 + (x2 - 5) / 100
```

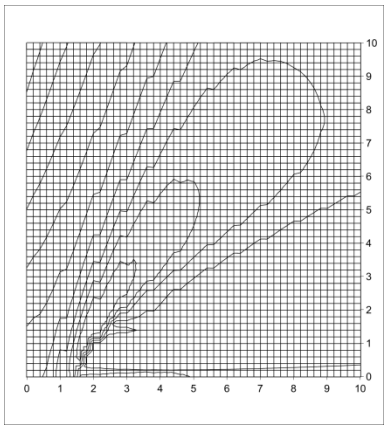
```
y_trueOld = 0
y_trueOldOld = 0
y_modelOld = 0
y_modelOldOld = 0
Sum = 0
For i_stage = 1 To 100
   If i_stage = 1 Then push = 5
   If i_stage = 20 Then push = 0
   If i_stage = 40 Then push = -2
   If i_stage = 60 Then push = 7
   If i_stage = 80 Then push = 3
   y_true = 0.3 * y_trueOld + 0.5 * y_trueOldOld + push
   y_model = x11 * y_modelOld + x22 * y_modelOldOld + push
   y_trueOldOld = y_trueOld
   y_trueOld = y_true
   y_modelOldOld = y_modelOld
   y_modelOld = y_model
   Sum = Sum + (y_true - y_model) ^ 2
Next i_stage
rms = Sum / (i_stage - 1)
f_of_x = 2 * Log(rms + 1)
```



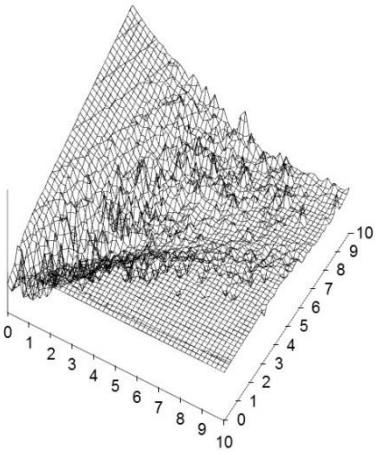
Algae Pond (#63) – This seeks to determine an economic optimization of an algae farm. Algae are grown in a pond, in a batch process, in wastewater from a fertilizer plant that contains needed nutrients. The algae biomass is initially seeded in the pond, grows naturally in the sunlight, and eventually is harvested. The questions are, "How deep should the pond be?" and "How long a time between harvestings?" Considering depth: A deeper pond has greater volume and can grow more algae. But sunlight attenuates with pond depth; so, a too deep pond does not add growing volume. Instead, a too deep pond creates more water that must be processed in harvesting, which increases costs. Considering growth time: Initially, there are no algae in the water. The initial seeded batch grows in time, but the mass asymptotically approaches a steady-state concentration when growth rate matches death rate. Growth is limited by CO2 infusion and sunlight, and high concentration of biomass shadows the lower levels. If you harvest on a short time schedule, you have a low algae concentration, produce low mass on a batch, but still have to process all the water. Alternately, if you harvest on a long time schedule, the excessive weeks don't contribute biomass, and you lose batches of production per year.

The simulation uses a simple model of the algae growth, and I appreciate Suresh Kumar Jayaraman's role in devising the simulator. The objective function is to maximize annual profit, which is based on value of the mass harvested per year less harvest costs and fixed costs.

The DVs are scaled to represent 2 to 25 batch growth days (horizontal) and 0.1 to 2 meters depth (vertical). The minimum is in the lower left of the contour, a steep valley. The feature in the lower right is a broad maximum that sends downhill searches either upward to the valley, or downward toward the axis representing a very shallow pond.



There are three difficulties with this problem. First, the steep valley causes difficulty for some gradient-based optimizers. Second, the nearly flat plane in the lower right can lead to premature convergence. Third, the surface has slope discontinuities due to the time discretization of the simulation. These can be made more visible with a larger simulation time increment, and appear as irregularities on the contour lines on the left. Ideally, the function is continuous in time, and the contours are smooth. Larger simulation discretization time intervals permit the simulation to run faster, which is desirable; but this has the undesired impact of creating discontinuities. Even with the small discretization used to generate the contours on the right, where there are no visible discontinuities, the surface undulations are present, and will confound gradient- and Hessian-based searches.



One can add a fourth difficulty, uncertainty on parameter and coefficient values in the model. Which creates a stochastic “live” fluctuations to the surface.

The VBA code is:

```
If x2 < 0 Or x1 < 0 Or x1 > 10 Or x2 > 10 Then
    constraint = "FAIL"
    Exit Function
End If
Dim C68(5000) As Single
Dim p68(5000) As Single
Dim n68(5000) As Single
Dim profit68 As Single
add_noise = Worksheets("Main").Cells(8, 12) * Sqr(-2 * Log(Rnd())) * Sin(2 * 3.14159 * Rnd())
Izero68 = 6 * (1 + add_noise)
add_noise = Worksheets("Main").Cells(8, 12) * Sqr(-2 * Log(Rnd())) * Sin(2 * 3.14159 * Rnd())
K168 = 0.01 * (1 + add_noise) 'growth rate constant
add_noise = Worksheets("Main").Cells(8, 12) * Sqr(-2 * Log(Rnd())) * Sin(2 * 3.14159 * Rnd())
K268 = 0.03 * (1 + add_noise) 'death rate constant
add_noise = Worksheets("Main").Cells(8, 12) * Sqr(-2 * Log(Rnd())) * Sin(2 * 3.14159 * Rnd())
alpha68 = 5 * (1 + add_noise)
add_noise = Worksheets("Main").Cells(8, 12) * Sqr(-2 * Log(Rnd())) * Sin(2 * 3.14159 * Rnd())
tr68 = 288 * (1 + add_noise) 'pond temperature
add_noise = Worksheets("Main").Cells(8, 12) * Sqr(-2 * Log(Rnd())) * Sin(2 * 3.14159 * Rnd())
topt68 = 293 * (1 + add_noise)   'optimum temperature
add_noise = Worksheets("Main").Cells(8, 12) * Sqr(-2 * Log(Rnd())) * Sin(2 * 3.14159 * Rnd())
kt68 = 0.001 * (1 + add_noise) 'temperature rate const
add_noise = Worksheets("Main").Cells(8, 12) * Sqr(-2 * Log(Rnd())) * Sin(2 * 3.14159 * Rnd())
kp68 = 0.02 * (1 + add_noise)  'P growth rate const
add_noise = Worksheets("Main").Cells(8, 12) * Sqr(-2 * Log(Rnd())) * Sin(2 * 3.14159 * Rnd())
```

```
    kn68 = 0.02 * (1 + add_noise)  'N growth rate const
    add_noise = Worksheets("Main").Cells(8, 12) * Sqr(-2 * Log(Rnd())) * Sin(2 * 3.14159 * Rnd())
    dt68 = 0.01 * (1 + add_noise)  'time increment (hr) 0.1 reveals discretization discontinuities
    simtime68 = 2 + x1 * (25 - 2) / 10      'time: min is 2 days and max is 25 days
    Depth68 = 0.1 + x2 * (2 - 0.1) / 10      'depth: min is 0.1 m and max is 2 m
    M68 = simtime68 / dt68
    rp68 = 0.025  'gP/gAlgae
    np68 = 0.01 'gN/gAlgae
    pc68 = 0.0001  'critical P conc
    nc68 = 0.0001  'critical N conc
    hn68 = 0.0145 'half saturation N conc
    hp68 = 0.0104 'half saturation P conc
    C68(1) = 0.1  'scaled value - starts at 1% of maximum possible
    p68(1) = 0.1  'initial P conc
    n68(1) = 0.1  'initial N conc
    For i68 = 1 To M68
      'dependence of the biomass growth on temperature f(T), intensity f(I), phosphorus f(P), Nitrogen
F(N)
      If (p68(i68) - pc68) > 0 Then
        fp68 = ((p68(i68) - pc68)) / (hp68 + (p68(i68) - pc68))
      Else
        fp68 = 0
      End If
      If (n68(i68) - nc68) > 0 Then
        fn68 = ((n68(i68) - nc68)) / (hn68 + (n68(i68) - nc68))
      Else
        fn68 = 0
      End If
      ft68 = Exp(-kt68 * (tr68 - topt68) ^ 2)
      fi68 = (Izero68 / (alpha68 * Depth68)) * (1 - Exp(-alpha68 * Depth68))
      p68(i68 + 1) = p68(i68) + dt68 * (kp68 * (-fi68 * ft68 * fp68 * fn68 * rp68) * C68(i68))
      If p68(i68 + 1) < 0 Then p68(i68 + 1) = 0
      n68(i68 + 1) = n68(i68) + dt68 * (kn68 * (-fi68 * ft68 * fp68 * fn68 * np68) * C68(i68))
      If n68(i68 + 1) < 0 Then n68(i68 + 1) = 0
      C68(i68 + 1) = C68(i68) + dt68 * (((fi68 * ft68 * fp68 * fn68) - K268 * C68(i68)) * C68(i68))
      If C68(i68 + 1) < 0 Then C68(i68 + 1) = 0
    Next i68
    avg68 = C68(i68 - 1)
    sales68 = 1 * avg68 * 25 * Depth68 * 365 / (simtime68 + 5)    '$/year    5 days to process pond
    expenses68 = 10 * Depth68 * 25 * 365 / (simtime68 + 5)          '$/year    cost to process and stock
    profit68 = sales68 - expenses68
    f_of_x = -profit68
    f_of_x = 10 * (f_of_x + 16700) / 43000   'for 25 days and 2 meters
```

Classroom Lecture Pattern (#73) – Lectures are effective in the beginning when students are paying attention. But, as time drags on, student attention wanders, and fewer and fewer students are sufficiently

intently focused to make the continued lecture be of any use. A "logistic" model that represents the average attention is:

$$f = \frac{1}{1 + e^{(t-15)}}$$

Where f is the fraction of class attention and t is the lecture time in minutes. Graph this, and you'll see that after 15 minutes f=0.5, half the class is not there. After about 20 minutes, effectively no one is attentive to the lecturer. The average f over a 55-minute period is about 0.26. Students only "get" 26% of the material. That is not very efficient. A teacher wants to maximize learning, as measured by maximizing the average f over a lecture. She plans on doing this by taking 3-minute breaks in which the students stand up, stretch, and slide over to the adjacent seat; then resume the lecture with fully renewed student attention (t=0 again). If a teacher lectured 2 minutes and took a three minute break, on a 5-minute cycle, then the value of f would be nearly 1.00 for the 2 minutes, but there would only be 2*(55/5)=22 minutes of lecture per session. But! 22 minutes at an f of 0.9999 over the 55 minutes is an average f of 0.40. This is an improvement in student focus on the lecture material. Is there a better lecture-stretch cycle period?

It can be argued that the lecture segments should have an equal duration. If there is an optimal duration, then exceeding it in one segment loses more than what is not lost in the other. There would be n equal duration lecture segments and (n-1) 3-min breaks within in a class period of p total minutes.

$$nx + (n-1)3 = p$$

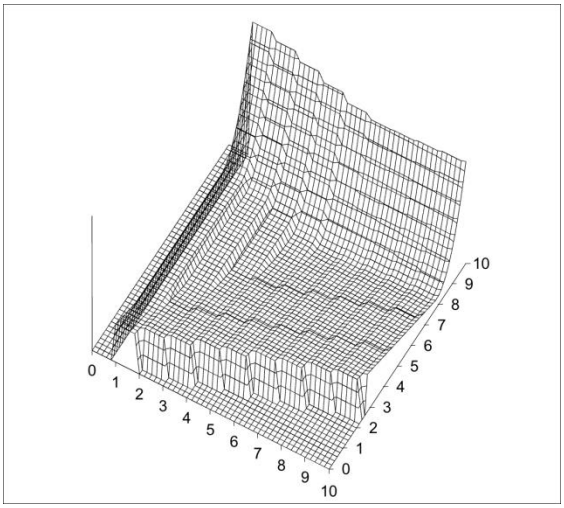
How long should a class period be to maximize learning? Say a school wants to have 7 lecture periods in a day so that students have a one-period lunch break and are in 6 classes each semester, and needs to permit 10 minutes for between class transitions. Longer classes would permit more learning. The work day length would be

$$l = 7p + (7-1) * 10$$

But an excessive day would receive objection from the faculty member's home. So, let's maximize learning by choosing p and n, but penalize an excessive day.

The code is:

```
n_segments = Int(1 * x1)    '# segments between 3-min breaks, integer value
Period = 10 * x2            'lecture duration, continuous, minutes
If n_segments > Period / 3 + 1 Or n_segments < 1 Then
        constraint = "FAIL"
        Exit Function
End If
Sum = 0     'initialize integral of student attention
For x11 = 0 To ((Period + 3) / n_segments - 3) Step 0.001
        Sum = Sum + 0.001 / (1 + Exp(x11 - 15))
Next x11
Duration = 7 * Period + 6 * 10      'work-day length planning 6 class periods per day
penalty = 0
If Duration > 10 * 60 Then penalty = 0.05 * (Duration - 10 * 60) ^ 2
f_of_x = -7 * n_segments * Sum + penalty
f_of_x = 10 * (f_of_x + 458) / (1617)
```



This is a mixed integer continuous DV problem. The front axis in the figure represents the number of lecture segments within a class period and the RHS axis represents classroom period in 10-minute intervals.

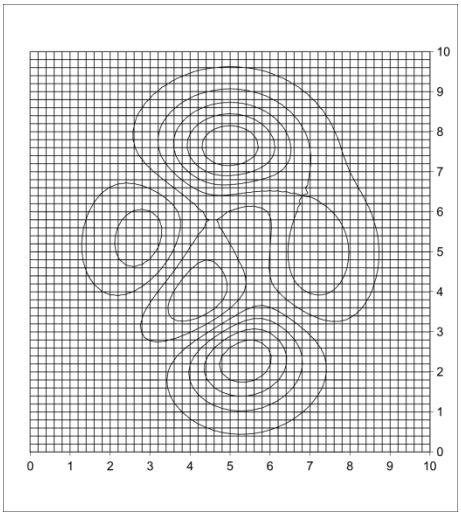
The horizontal direction in each segment is flat, and becomes discontinuous when the n integer value changes.

Not visible in this view, is the additional problem of the time discretization in using the rectangle rule of integration to solve for the average f-value. With a time step of .001 minutes, it is not visible. But with a step size of 1 minute, the discontinuity along the "continuous" "p" axis becomes visible.

Mountain Path (#29) – This uses the Peaks function to represent the land contour of a mountain and valley region. City A is at the location (1,1) and city B at the location (9,9). The objective is to find a path from A to B that minimizes distance, and avoids excessive steepness along the path. The path is described as a cubic relation of y (the vertical axis in the Peaks contour) and x (the horizontal axis in the Peaks contour).

$$y = a + bx + cx^2 + dx^3$$

If the path follows a straight line on the x-y projection surface, from (1,1) the path goes over the mountain at (4,4), down the valley at (5.5,5.5), over the side of the mountain at (7,7) and down to (9,9). Alternately, if the path goes from (1,1) to (5,3) it stays on level ground. Then to (7, 6.5) it does not go into the middle valley, and only rises to the lowest point (the pass) between the two upper-right mountains. While the projection of the curved path might be longer, it does not rise or fall, and is actually a shorter path.



A section of an alternate path might go from (6,6) to (8,7) taking the shortest distance across the mountain pass, but this might have a steep climb at about (6.5, 6.5). An alternate path through the pass, from (6, 4) to (8,8), sideways up the mountain would not have the steep climb.

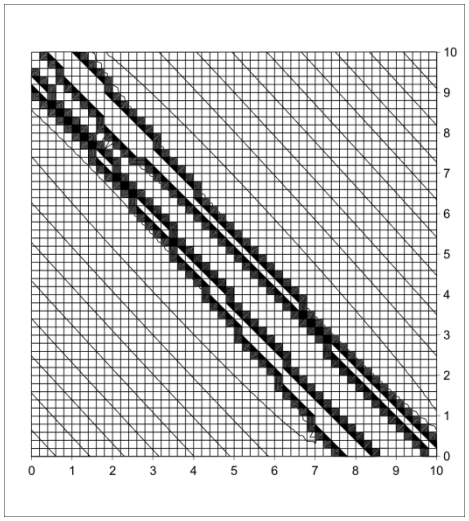
In the Peaks function the DVs are the x and y direction, and the objective is to find the min (or max) point. By contrast, here, the DVs are coefficients c and d in the y(x) relation. Once the optimizer chooses trial values for c and d, a and b are determined by the equality constraints of y(at xA)=yA, and y(at xB)=yB. The path length includes the x, y, and z (elevation) distances and the function calculates path length by discretizing the path into 100 Δx increments, and summing the 100 $\Delta s = \sqrt{\Delta x^2 + \Delta y^2 + \Delta z^2}$ elements. The steepness of the path is evaluated at each of the 100 increments and if $\left|\frac{dz}{ds}\right|$ exceeds a threshold, the constraint is violated

The objective statement is:

$$\underset{\{c,d\}}{min} J = S = \sum \Delta s$$

$$S.T. \left|\frac{dz}{ds}\right| \leq slope$$

The function S(c,d) results in multiple local optima comprised of steep, narrow valleys with low sloping floor. The dark bands indicate steep rises to a maximum value at constrained conditions. The minimum is in the (DV1=7, DV2=2) area. However, if trial solutions start outside of that area, constraints would prevent the trial solution form migrating to the optimum. Many optimizers migrate to local optima at about (4,4) and (5,6).



The VBA code is:

```
  CoeffC = 0.2 * x1 - 1.8 'scaled for best reveal of the function detail
  CoeffD = 0.012 * x2 + 0
  xa = 1          'start location - E-W direction
  ya = 1          'start location - N-S direction
  xb = 9          'end location
  yb = 9
  CoeffB = ((yb - ya) - CoeffC * (xb ^ 2 - xa ^ 2) - CoeffD * (xb ^ 3 - xa ^ 3)) / (xb - xa)
  CoeffA = ya - CoeffB * xa - CoeffC * xa ^ 2 - CoeffD * xa ^ 3
  N = 100
  deltax = (xb - xa) / N
  PathS = 0
  MaxSteepness = 0
  For xstep = 1 To N
    x = xa + xstep * deltax
    y = CoeffA + CoeffB * x + CoeffC * x ^ 2 + CoeffD * x ^ 3
    dydx = CoeffB + 2 * CoeffC * x + 3 * CoeffD * x ^ 2 'derivative of y w.r.t. x
    x11 = 3 * (x - 5) / 5     'convert my 0-10 DVs to the -3 to +3 range for the function
    x22 = 3 * (y - 5) / 5
    zbase = 3 * ((1 - x11) ^ 2) * Exp(-1 * x11 ^ 2 - (x22 + 1) ^ 2) - _
      10 * (x11 / 5 - x11 ^ 3 - x22 ^ 5) * Exp(-1 * x11 ^ 2 - x22 ^ 2) - _
      (Exp(-1 * (x11 + 1) ^ 2 - x22 ^ 2)) / 3
    zbase = (zbase + 6.75) / 1.5
    x11 = 3 * (x + 0.0001 - 5) / 5   'convert my 0-10 DVs to the -3 to +3 range for the function
    Z = 3 * ((1 - x11) ^ 2) * Exp(-1 * x11 ^ 2 - (x22 + 1) ^ 2) - _
      10 * (x11 / 5 - x11 ^ 3 - x22 ^ 5) * Exp(-1 * x11 ^ 2 - x22 ^ 2) - _
      (Exp(-1 * (x11 + 1) ^ 2 - x22 ^ 2)) / 3
    Z = (Z + 6.75) / 1.5
    pzpx = (Z - zbase) / 0.0001     'partial derivative of elevation w.r.t. x
    x11 = 3 * (x - 5) / 5     'convert my 0-10 DVs to the -3 to +3 range for the function
    x22 = 3 * (y + 0.0001 - 5) / 5
```

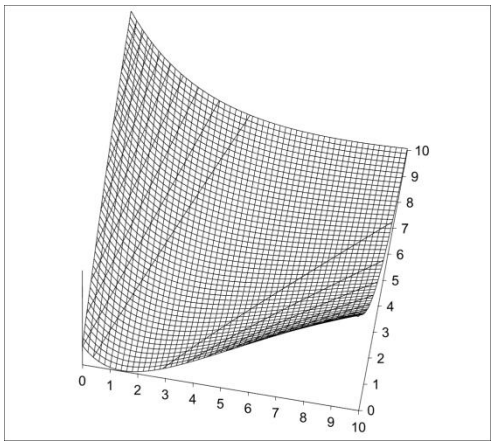
```
    Z = 3 * ((1 - x11) ^ 2) * Exp(-1 * x11 ^ 2 - (x22 + 1) ^ 2) - _
      10 * (x11 / 5 - x11 ^ 3 - x22 ^ 5) * Exp(-1 * x11 ^ 2 - x22 ^ 2) - _
      (Exp(-1 * (x11 + 1) ^ 2 - x22 ^ 2)) / 3
    Z = (Z + 6.75) / 1.5
    pzpy = (Z - zbase) / 0.0001     'partial derivative of elevation w.r.t. y
    dzdx = pzpx + pzpy * dydx       'total derivative of elevation w.r.t. x
    dsdx = Sqr(1 + dydx ^ 2 + dzdx ^ 2) 'total derivative of path length w.r.t. x
    dPathS = deltax * dsdx      'Path segment length
    PathS = PathS + dPathS      'Cumulative path length
    dzds = dzdx / dsdx          'Path slope
    Steepness = Abs(dzds)
    If Steepness > MaxSteepness Then MaxSteepness = Steepness
    badness = 0
    '   Soft Constraint
'    If Steepness > 0.8 Then    '   .85=tan(40),  .60=tan(30),   .35=tan(20
'       badness = (Steepness - 0.8)
'       PathS = PathS + 2 * badness ^ 2
'    End If
    '   Hard Constraint
    If Steepness > 0.95 Then   '   .85=tan(40),  .60=tan(30),   .35=tan(20)
      constraint = "FAIL"
      f_of_x = 10           ' visually acknowledge violation on contour
      Exit Function
    End If
  Next xstep
  f_of_x = PathS
  f_of_x = 10 * (PathS - 13.28) / 52.36
```

Reaction with Phase Equilibrium (#75) – This has a reversible homogeneous reaction of A + B goes to C; but, there are two immiscible phases, oil and water, and the A, B, and C species are dispersed in both. The A and B species are preferentially in water, and C is preferentially soluble in the oil. The reaction is slow relative to mass transfer and diffusion, so the A, B, and C species are considered in thermodynamic equilibrium between the oil and water phases.

1. The reaction in the oil layer of volume v is a+b↔c, r=-k1ab+k2c. Lower case letters represent oil.
2. In the water layer of volume V is A+B↔C, R=-k3AB+k4C. Upper case letters represent water.
3. Each component is in concentration equilibrium between the two phases KA=a/A, KB=b/B, KC=c/C
4. The total number of moles of each species are NA (=va+VA), NB, and NC.
5. Given an initial number of moles, NAo, and phase equilibria, ao=NAo/(v+V/KA) and Ao=NAo/(vKA+V). bo, co, Bo, Co are similarly calculated.
6. Mass balances on species a and A in the individual oil and water phases, added, disappear the unknowable interphase rates, and result in

$$\frac{dNA}{dt} = vr + VR = -vk1ab + vk2c - Vk3AB + Vk4C$$

7. Equilibrium relates the a and A concentrations to NA as in line 5. Stoichiometry and equilibrium relate the b, c, B, C concentrations to NA. Concentrations a, b, c, A, B, and C can be reformulated in terms of NA. Stoichiometry gives: NB=NBo-(NA-NAo). Equilibrium gives: b=NB/(v+V/KB). Combined: (NBo-(NA-NAo)) /(v+V/KB). Etc.
8. There are 4 rate coefficients (k1, k2, k3, k4); however, only two are independent. The reaction is the same whether in the oil phase or the water phase. But, the reaction is not really dependent on concentrations. It is dependent on activities of the species in the oil and water medium. Further, the equilibrium of a to A (b to B, and c to C) are defined as equal activities. This means that the activity coefficients are related to the partition coefficient. KA=γa/γA. Etc. Then k3=k1*KA*KB, and k4=k2*KC. So,
$$\frac{dNA}{dt} = vr + VR = -vk1ab + vk2c - V(k1KAKB)AB + V(k2KC)C$$
9. This differential equation represents a phase-equilibrium constrained model.
10. Only the one nonlinear ODE needs to be solved, because all species concentrations are related to the single NA and the initial conditions on NAo, NBo, and NCo.
11. This can be expressed in terms of reaction extent.
$$\xi = NAo - NA$$
$$\frac{d\xi}{dt} = vk1ab - vk2c + V(k1KAKB)AB - V(k2KC)C$$
12. Which is solved with a simple Euler's method.



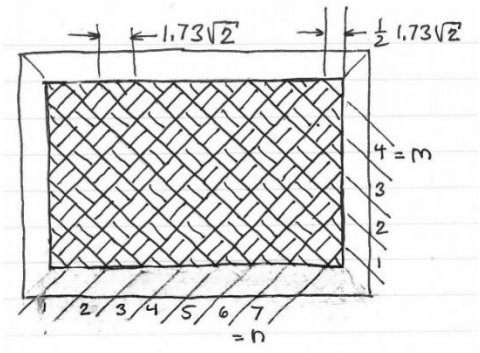
The optimization seeks to find k1 and k2 values that make the dynamic model best match the experimental data in a least squares sense. The function has the "steep valley" property that makes many optimizes converge along the bottom, with parameter correlation.

The VBA Code is:

```
    k1o75 = 200 + 200 * (x1 - 5) / 5          'oil forward reaction rate coefficient
    k2o75 = 0.015 + 0.015 * (x2 - 5) / 5    'oil backward reaction rate coefficient
    If k1o75 < 0 Or k2o75 < 0 Then
      constraint = "FAIL"
      Exit Function
    End If
    sum75 = 0      'sum of penalties for deviations from measured values
    dt75 = 0.5     'time increment for Euler's method, min
    vo75 = 100     'volume of oil, liters, human supposition
    VW75 = 200     'volume of Water, liters, human supposition
    NA75 = 5       'initial moles of A, combined, in oil and water, human supposition
    NB75 = 3       'initial moles of B, human supposition
    NC75 = 1       'initial moles of C, human supposition
    KA75 = 0.1     'partition coefficient for A, human supposition
    KB75 = 0.05    'for B, human supposition
    KC75 = 5.5     'for C, human supposition
    k1W75 = k1o75 * KA75 * KB75    'water forward reaction rate coefficient
    k2W75 = k2o75 * KC75           'water backward reaction rate coefficient
    e75 = 0                        'reaction extent sum of oil and water
```

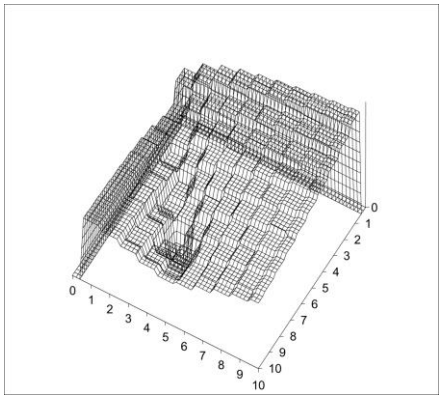
```
For i75 = 1 To 80            '80 time steps for dynamic model
   ao75 = (NA75 - e75) / (vo75 + VW75 / KA75)  'concentration of A in oil
   bo75 = (NB75 - e75) / (vo75 + VW75 / KB75)  'B in oil
   co75 = (NC75 + e75) / (vo75 + VW75 / KC75)  'C in oil
   AW75 = (NA75 - e75) / (vo75 * KA75 + VW75)  'A in water
   BW75 = (NB75 - e75) / (vo75 * KB75 + VW75)  'B in water
   CW75 = (NC75 + e75) / (vo75 * KC75 + VW75)  'C in water
   ro75 = k1o75 * ao75 * bo75 - k2o75 * co75   'reaction rate in oil
   RW75 = k1W75 * AW75 * BW75 - k2W75 * CW75   'reaction rate in water
   e75 = e75 + dt75 * (vo75 * ro75 + VW75 * RW75)          'Euler's method
   t75 = i75 * dt75                                        'simulation time
                'data generated by Excel simulator - "Kinetic Model with Phase Equilibrium Sheet 1"
   If i75 = 8 Then sum75 = sum75 + (e75 - 0.235176) ^ 2      'penalty for deviation from data
   If i75 = 16 Then sum75 = sum75 + (e75 - 0.361498) ^ 2
   If i75 = 24 Then sum75 = sum75 + (e75 - 0.453315) ^ 2
   If i75 = 32 Then sum75 = sum75 + (e75 - 0.590891) ^ 2
   If i75 = 40 Then sum75 = sum75 + (e75 - 0.658838) ^ 2
   If i75 = 48 Then sum75 = sum75 + (e75 - 0.637532) ^ 2
   If i75 = 56 Then sum75 = sum75 + (e75 - 0.717511) ^ 2
   If i75 = 64 Then sum75 = sum75 + (e75 - 0.701781) ^ 2
   If i75 = 72 Then sum75 = sum75 + (e75 - 0.70742) ^ 2
   If i75 = 80 Then sum75 = sum75 + (e75 - 0.763436) ^ 2
Next i75
f_of_x = 10 * (sum75 - 0.0059) / 18
```

Cork Board (#1) – This seeks to determine the number of rows and columns for a bulletin board made of wine corks. The length of a cork is about 1.73 inches, which is nearly double the width. So a pair of adjacent corks makes a square, and squares tile a surface. Artistically, the aspect ratio of a picture (the cork region) should be equal to the golden ratio, $\gamma = \frac{-1+\sqrt{5}}{2} \cong 0.61803$.



Functionally, it needs a minimum of about 150 corks to provide adequate posting area, but more than that takes assembly time and uses up an endangered commodity (I only use real corks, which are being replaced with plastic stoppers or screw tops). There are many ways to tile the corks – rectangular, chevron, etc. and aligned with the frame or on the diagonal. The illustrations show rectangular on the diagonal. In any pattern, there are an integer number of rows and columns, so the dimensions depend on the integer. The DVs are the number of rows

and columns, and the OF is a combination of the deviation from the ideal γ and 150 corks. In the combined OF I choose EC factors of 0.05 deviation from the golden ratio as creating concern equivalent to 50 corks deviation from the target.

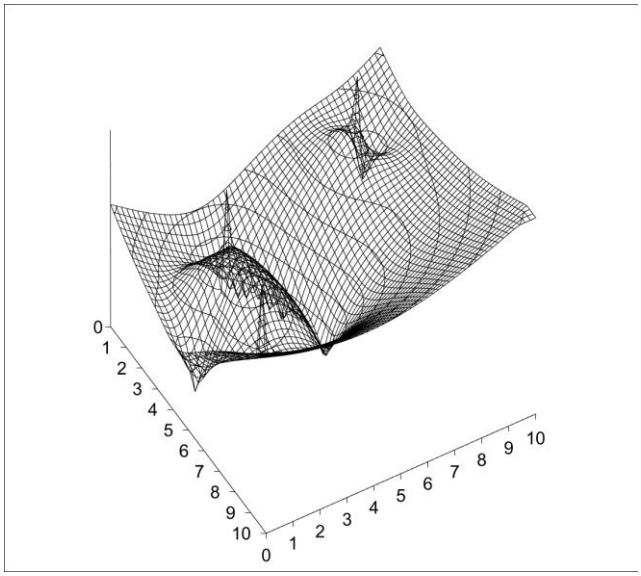


This has integer DVs, which creates flat surfaces with discontinuities between. While there is a general trend to the minimum, there are local traps (minima surrounded by worse spots). Far from the minimum, the large OF values make the entire minimum region seem featureless. So to better visualize the region of interest, I log transformed the OF.

The VBA Code is:

```
  If Fchoice = 1 Then  'Cork Board dimension
    n = Int(x1 + 0.5)   'number of full rows on the bias for length count
    m = Int(x2 + 0.5)   'number of full rows on the bias for height count
    If n < 1 Or m < 1 Then
      constraint = "FAIL"
      Exit Function
    End If
    corks = 4 * (n + 1) * (m + 1)   'number of corks needed
    L1 = (n + 1) * Sqr(2) * 1.73 '+ 2 * 2.5     'frame inside length add 2 times thickness for exterior length
    L2 = (m + 1) * Sqr(2) * 1.73 '+ 2 * 2.5     'frame inside height
    ratio = L2 / L1
    golden = (-1 + Sqr(5)) / 2      'ideal aspect ratio
    badness1 = (ratio - golden) ^ 2     'deviation from ideal
    badness2 = (150 - corks) ^ 2       'deviation from ideal
    ec1 = 0.05                    'Equal Concern factor
    ec2 = 50                      'Equal Concern factor
    f_of_x = badness1 / ec1 ^ 2 + badness2 / ec2 ^ 2    'the OF
    f_of_x = Log(f_of_x)                'transform to make the features in low values more visible
    f_of_x = 10 * (f_of_x + 2.83) / 12      'scaling on a 0 to 10 basis
    Exit Function
  End If
```

Three Springs (#12) – In this three springs are joined at a common, but floating Node 4. They are fixed at the other ends Nodes 1, 2, and 3. Initially all springs are at their resting length. There is no tension or compression in any spring. Then Node 3 is moved. This seeks to determine the location of the common Node 4 that balances the x- and y-force on Node 4. Springs are linear and compression and tension have the same k. Springs are not bendable. Interestingly, there are several unstable equilibria locations. The code has an option to minimize energy in the springs. Shown is the force balance version.



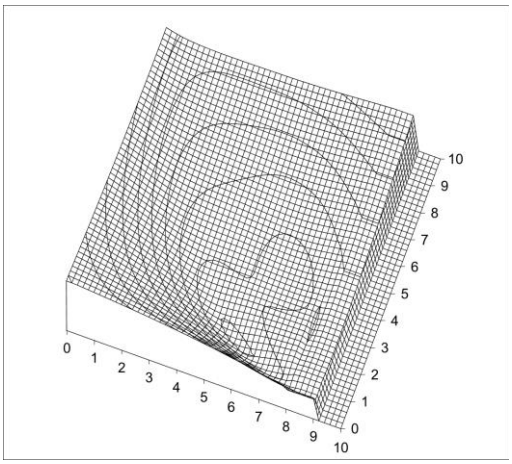
There are spikes at Nodes 1, 2, and 3. If Node 4 is at any of those points, then the compression of a spring is infinity. Thanks to MAE MS Candidate Romit Maulik for the idea, summer 2015, who was actually seeking to minimize the collective energy in all springs.

The VBA Code is:

```
If Fchoice = 12 Then        'Three Springs, unbendable, linear
'  Thanks to Romit Maulik for the basic idea - June 2015
'  All three springs are joined at the common, and free to move (y4, x4) node
'  Originally one end of Spring 1 is fixed at position (y1,x1), Spring 2 at (y2,x2), and 3 at (y3o, x3o)
'  Originally all three springs are at rest
'  Node 3 moves to (y3, x3).  Where does Node 4 move to?
'  OF from Case 1:  At rest the y and x forces on Node 4 must be balanced. Then
'     minimize the squared sum of component forces.  This works, but permits
'     unstable equilibrium points of high tension when the forces are balanced.
'  OF from Case 2:  A more rational surface happens when finding the Node 4 position
'     that minimizes the energy in all springs.
   F12y1 = 8   'you can initialize springs in any locations
   F12x1 = 2
   F12y2 = 2
   F12x2 = 4
   F12y3o = 7
   F12x3o = 7
   F12y4o = 5
   F12x4o = 5
   F12y3 = 1   'choose a variety of interesting move to points
   F12x3 = 8
   F12y4 = x2
   F12x4 = x1
   F12k1 = 1   'choose a variety of values for the spring constants
   F12k2 = 2
   F12k3 = 0.05
   F12L1o = Sqr((F12y1 - F12y4o) ^ 2 + (F12x1 - F12x4o) ^ 2)   'spring original at rest lengths
```

```
   F12L2o = Sqr((F12y2 - F12y4o) ^ 2 + (F12x2 - F12x4o) ^ 2)
   F12L3o = Sqr((F12y3o - F12y4o) ^ 2 + (F12x3o - F12x4o) ^ 2)
   F12L1 = Sqr((F12y1 - F12y4) ^ 2 + (F12x1 - F12x4) ^ 2)      'spring lengths after Node 3 moves
   F12L2 = Sqr((F12y2 - F12y4) ^ 2 + (F12x2 - F12x4) ^ 2)
   F12L3 = Sqr((F12y3 - F12y4) ^ 2 + (F12x3 - F12x4) ^ 2)
'  Force Analysis (more interesting surface, permits unstable equilibrium)
   If F12L1 = 0 Or F12L2 = 0 Or F12L3 = 0 Then  'to prevent a divide by zero
      constraint = "FAIL"
      f_of_x = 10
      Exit Function
   End If
   F12F1 = F12k1 * (F12L1o - F12L1)    'negative if tension, positive if compression
   F12F2 = F12k2 * (F12L2o - F12L2)
   F12F3 = F12k3 * (F12L3o - F12L3)
   F12F1y = F12F1 * (F12y4 - F12y1) / F12L1
   F12F1x = F12F1 * (F12x4 - F12x1) / F12L1
   F12F2y = F12F2 * (F12y4 - F12y2) / F12L2
   F12F2x = F12F2 * (F12x4 - F12x2) / F12L2
   F12F3y = F12F3 * (F12y4 - F12y3) / F12L3
   F12F3x = F12F3 * (F12x4 - F12x3) / F12L3
   F12Fy = F12F1y + F12F2y + F12F3y
   F12Fx = F12F1x + F12F2x + F12F3x
   f_of_x = F12Fy ^ 2 + F12Fx ^ 2 + add_noise
   f_of_x = 2.5 * f_of_x ^ 0.25        'this power scaling emphasizes visibility of features in the low OF
region
''  Energy Analysis (more rational surface, and simpler to compute)
'   F12E1 = F12k1 * (F12L1o - F12L1) ^ 2
'   F12E2 = F12k2 * (F12L2o - F12L2) ^ 2
'   F12E3 = F12k3 * (F12L3o - F12L3) ^ 2
'   f_of_x = F12E1 + F12E2 + F12E3
'   f_of_x = Sqr(f_of_x)
   Exit Function
End If
```

Retirement (#15) – This investigates two decisions: when to retire and what portion of your salary you should set aside each year while working to finance retirement.

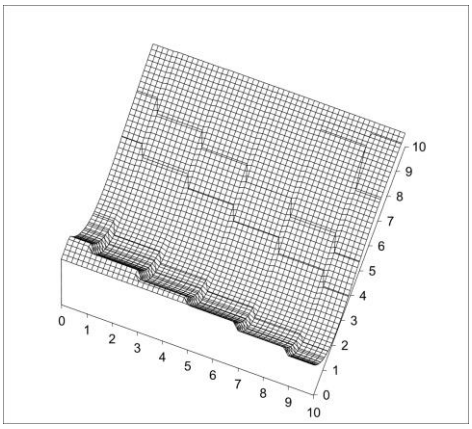


The objective is to maximize cumulative joy in life. Joy is not income. Joy is based on income, but is as much determined by the discretionary time that permits one to pursue personal happiness and by the affirmation pleasure of working.

The model has one start working at age 25, and die at 85. While working, a portion of the salary is invested in savings for retirement. In retirement, withdrawing from the savings provides income. But, since the person might live to 90, the

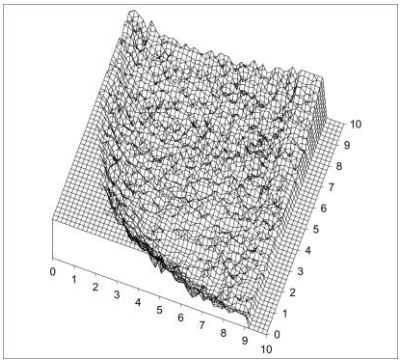
retirement income is allocated to last until age 90. One DV is the age to retire: Retiring at an early age would leave a long retirement period to pursue individual choices, but there would not be much $ saved to finance the pursuit. Retiring at a late age would mean that a lot of $ had accumulated to finance retirement activities, but then there is little time left to enjoy it. The other DV is the portion of salary to invest: Don't invest much, and there is more $ to enjoy life while working, but little in retirement. Invest a high portion of work-salary and there will be more $ to enjoy retirement, but it means a low-joy life while working. If retirement income is too high, then it is excessive to support joy, and means that the tax rate is very high.

The code accounts for the time value of $, and the compounding of investment $. The code also accounts for tax rate, and has joy as a diminishing returns consequence of salary. Further the code accounts for an age factor that diminishes the ability to enjoy with increasing age. As complicated as that seems, it is still a simple representation, not accounting for partner activity, pensions, inheritances, post-retirement employment, etc. I modeled this as how I perceive life with 50% of my time at work leading to personal joys (a professor's view) which also has a salary-capped academic position, a 50 percentile death age of 85, and an age of half-function at 80. But, you will make alternate career choices, and have differing life expectancy.



The 3-D response is shown above. DV1 is scaled age, from 40 to 90, and is on the lower-left axis. DV2 is scaled portion of salary, from 0 to 100%, on the right-hand axis. The discontinuity of the surface at DV1=8 represents the impact of attempting to work after the age factor has diminished and led to forced retirement.

The surface seems continuous, smooth, but a detailed look in the figure to the left, over the 67 to 72 year period, reveals the impact of age discretization



In addition, the nominal values of interest rate, inflation rate, half age and death age are uncertain. If selected these can be a stochastic factor.

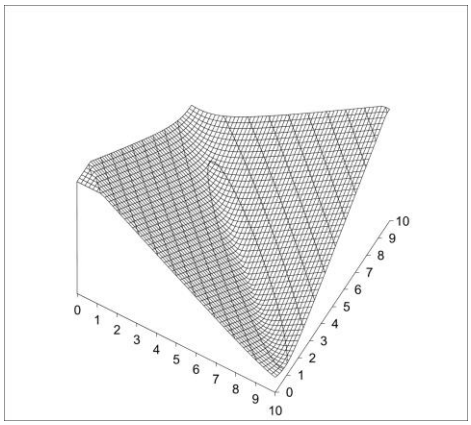
The VBA Code is:

```
If Fchoice = 15 Then
'  Seeks optimum retirement age and portion of annual salary to
'     set aside for retirement.
'  The discretization associated with year-to-year age of retirement makes
'     discontinuities that sibstantially confound gradient based techniques.
```

```
   salary15begin = 12
   salary15end = 150
   basesalary15 = salary15begin
   savings15 = 0
   joy15 = 0
   discount15 = 0.0375
   interest15 = 0.07
   halfage15 = 80
   deathage15 = 90
   plandeathage15 = 85
   retireage15 = Int(40 + x1 * 5 + 0.5)
'   retireage15 = Int(66 + 0.5 * x1 + 0.5)    'to see discretization detail near optimum
   portion15 = x2 / 10
'   portion15 = 0.06 + x2 / 40        'to see discretization detail near optimum

'   Stochastic?  If Yes, uncomment these lines
'    discount15 = discount15 + 0.002 * Sqr(-2 * Log(Rnd())) * Sin(2 * 3.14159 * Rnd()) '0.02 * (Rnd() - 0.5)
'    interest15 = interest15 + 0.002 * Sqr(-2 * Log(Rnd())) * Sin(2 * 3.14159 * Rnd()) '0.02 * (Rnd() - 0.5)
'    halfage15 = halfage15 + Int(2 * Sqr(-2 * Log(Rnd())) * Sin(2 * 3.14159 * Rnd()))  'Int(20 * (Rnd() - 0.5))
'    deathage15 = deathage15 + Int(2 * Sqr(-2 * Log(Rnd())) * Sin(2 * 3.14159 * Rnd()))   ' Int(30 * (Rnd() -
0.5))

   If retireage15 > plandeathage15 Then
      constraint = "FAIL"
'       f_of_x = 10
      Exit Function
   End If

   If retireage15 < plandeathage15 - 5 Then
      withdrawportion15 = 1 / (plandeathage15 + 5 - retireage15)
   Else
      withdrawportion15 = 0.2
   End If

   For age15 = 25 To deathage15 Step 1     'calculate cumulative joy over the lifetime
      agefactor15 = 0.1 + 0.9 / (1 + Exp(0.2 * (age15 - halfage15)))    'functionality to work or enjoy
      If agefactor15 < 0.5 And retireage15 > age15 Then retireage15 = age15 'forced retirement
      If age15 = retireage15 Then capitalwithdraw15 = savings15 * withdrawportion15
      If age15 <= retireage15 Then        'working
         savings15 = savings15 * (1 + interest15) + portion15 * salary15   'compound retirement savings15
         basesalary15 = basesalary15 * (1 + 0.15 * (salary15end - basesalary15) / (salary15end -
salary15begin)) 'my model for how my salary increased over my career
         salary15 = agefactor15 * basesalary15     'The base salary is for a 100% functioning person.  The
actual income could be reduced by days on the job, etc.
         discountedsalary15 = salary15 / (1 + discount15) ^ (age15 - 25) 'normalized to the start, at age 25,
by the inflation rate
         tax15 = 0.85 / (1 + Exp(0.1 * (35 - discountedsalary15 * (1 - portion15))))  'an approximate model
for the graduated income tax, based on income after investing in tax-deferred savings15
```

```
        joyfactor15 = 1 - 2.5 * Exp(-discountedsalary15 * (1 - portion15) * (1 - tax15) / 10) 'my approximation for how joy scales with discretionary $
        joy15 = joy15 + joyfactor15 * agefactor15 * (1 * 52 + 3 * 5) / 364 'joy of personal time per year - based on a 52 week year of 364 days
        joyfactor15 = 1 - 2.5 * Exp(-discountedsalary15 / 10)        'now based on salary as a measure of value to humanity
        joy15 = joy15 + joyfactor15 * agefactor15 * 0.5 * (5 * 45) / 364   'joy of the job per year - only 20% of the time generates joy on average.  My 50% of job is joy might not match most jobs.
      Else       'retired
        salary15 = capitalwithdraw15 + savings15 * interest15   'take all interest plus capital draw from savings15
        savings15 = (savings15 - salary15) * (1 + interest15)   'residual $ in savings15 compounds
        discountedsalary15 = salary15 / (1 + discount15) ^ (age15 - 25)
        tax15 = 0.85 / (1 + Exp(0.1 * (35 - discountedsalary15)))
        joyfactor15 = 1 - 2.5 * Exp(-discountedsalary15 * (1 - tax15) / 10)
        joy15 = joy15 + joyfactor15 * agefactor15 * (6 * 52) / 364  'joy  of personal time less one day per week for bills, taxes etc.
      End If
    Next age15
    age15 = age15 - 1
    If savings15 > 0 Then
      discountedsavings15 = savings15 / (1 + discount15) ^ (age15 - 25)
      joyfactor15 = 1 - 2.5 * Exp(-discountedsavings15 * (1 - tax15) / 10)
      joy15 = joy15 + joyfactor15
    Else
      joy15 = joy15 - 3
    End If
'    f_of_x = 10 * (-joy15 + 20) / 70 'normal scaling - misses floor detail
'    f_of_x = 10 * (-joy15 + 20) / 20 'normal scaling - sees floor detail but misses high detail
    f_of_x = 10 * Sqr(Sqr((-joy15 + 20) / 20)) - 3.5  'normal, transformed to reveal
'    f_of_x = 10 * (-joy15 + 20) / 20                 'for stochastic function
'    f_of_x = 10 * (-joy15 + 19.4) / 4              'to see discretization detail
    Exit Function
End If
```

Solving an ODE (#23) – Some differential equations can be solved exactly analytically. But, some are too complicated. This investigates the use of a polynomial relation to approximate the solution. Consider the ODE $\frac{dy}{dx} = f(x, y)$, $y(x_0) = y_0$. If there is a solution, $y(x)$, then it can be expanded in a Taylor series, and terms rearranged to make it a power series, $y(x) = a + bx + cx^2 + dx^3 + \cdots$. If the truncated power series (for example a cubic) is a reasonable approximation, then its derivative, $\frac{dy}{dx} = b + 2cx + 3cx^2$, is a reasonable approximation to $\frac{dy}{dx} = f(x, y) \cong f(x, a + bx + cx^2 + dx^3) = b + 2cx + 3cx^2$ for all x-values. The objective is



to find the b and c values that makes $b + 2cx + 3cx^2$ best match $f(x, a + bx + cx^2 + dx^3)$ for many x-values.

With a quadratic approximation (appropriate for a 2-D exercise), and b and c chosen by the optimizer, the value of coefficient "a" is determined from the initial condition, $y_0 = a + bx_0 + cx_0{}^2$. The generic optimization statement is:

$$\min_{\{b,c\}} J = \sum_{i=1}^{N} [f(x, a + bx + cx^2) - (b + 2cx)]^2$$

S.T. $\Delta x = (x_{end} - x_0)/N$
$x_i = x_0 + i\Delta x$
$a = y_0 - bx_0 - cx_0{}^2$

My 2-D Example uses the quadratic polynomial to approximate the solution to a first order linear ODE $\tau \frac{dy}{dx} + y = k,\ y(x_0) = y_0$. Where $f(x, y) = (k - y)/\tau$. The specific optimization statement is:

$$\min_{\{b,c\}} J = \sum_{i=1}^{N} \left[\frac{k - (a + bx + cx^2)}{\tau} - (b + 2cx) \right]^2$$

S.T. $\Delta x = (x_{end} - x_0)/N$
$x_i = x_0 + i\Delta x$
$a = y_0 - bx_0 - cx_0{}^2$

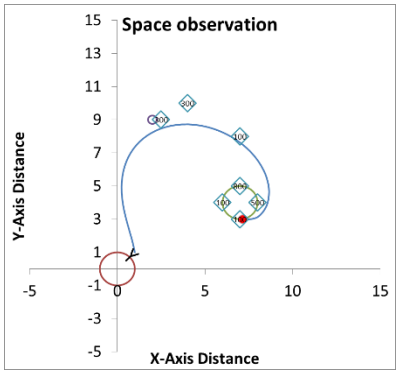
The function is a steep valley type.

The VBA Code is:

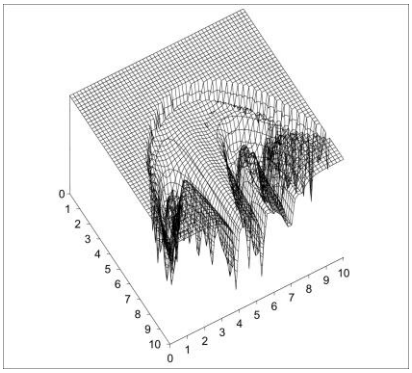
```
If Fchoice = 23 Then            'RRR power series model solution to an ODE
'  Best fit of a polynomial y=a+bx+cx^2 as a solution to the
'  linear first order ODE   tau*(dy/dx)+y=k
'  By minimizing the difference between the polynomial derivative =b+2cx and
'  The ODE derivative = (k-y)/tau
'   If x1 < 0 Or x2 < 0 Or x1 > 10 Or x2 > 10 Then
'      constraint = "Fail"
'      Exit Function
'   End If
   x023 = 1
   xend23 = 10
   y023 = 1
   k23 = 10
   tau23 = 4
   b23 = 2 * (x1 - 5)
   c23 = 0.2 * (x2 - 5)
   a23 = y023 - b23 * x023 - c23 * (x023) ^ 2
   sum23 = 0
   deltax23 = (xend23 - x023) / 100
   For i23 = 1 To 100
```

```
    x23 = x023 + i23 * deltax23
    y23 = a23 + b23 * x23 + c23 * (x23) ^ 2
    sum23 = sum23 + (((k23 - y23) - tau23 * (b23 + 2 * c23 * x23)) * deltax23) ^ 2
  Next i23
  f_of_x = 10 * (Sqr(sum23) - 2) / 150
  Exit Function
End If
```

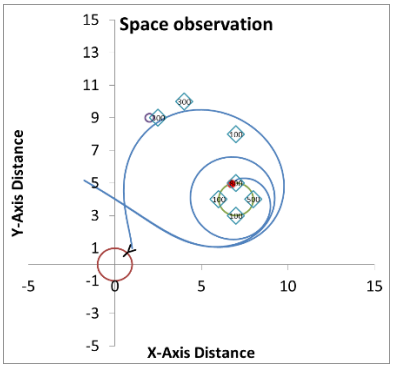
Angry Birds Space (#26) – Determine 1) the drawback on the slingshot and 2) the angle to aim it, and send the bird along its free-fall trajectory in space to accumulate piggy points.



In this example there are three planets. The home planet is at the origin and the main piggy-planet is centered at 4-up and 7-over. The third, smaller planet is at 9-up, 2-ver. All three planets have gravitational pull on the bird, but remain stationary. The two larger planets have an atmosphere that adds drag to the bird when in the proximity. Otherwise translational x- and y-motion is governed by the gravitational force balance in this 2-D space.

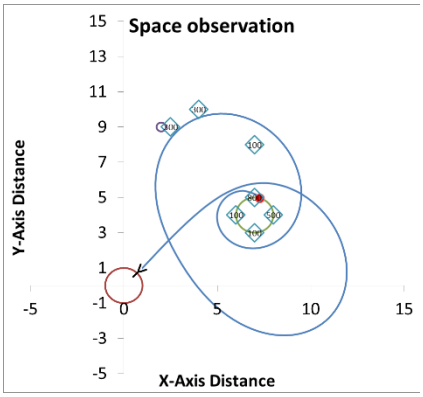


The diamond markers indicate piggy out-posts in space, and larger ones on the planets. Get near or crash into them to gather piggy points. I have a separate Excel/VBA program to visualize the bird projectile path through space. The OF is the piggy-points accumulated and the DVs are % and angle. There are many local optima, and very narrow pinholes. There are flat spots. It was a surprisingly difficult surface for optimization.



One of the best paths sends the bird upward, gravity pulls it to the right passing near the pigs-in-space points, then overshoots the piggy planet but is prevented from going into deep space by the combined gravitational pull (at about 5 on the vertical axis). It then falls back toward the piggy planet, encircles it, and eventually crashes into the north pole piggy post.

There are many nearly as piggy-point productive patterns.



The VBA Code is:

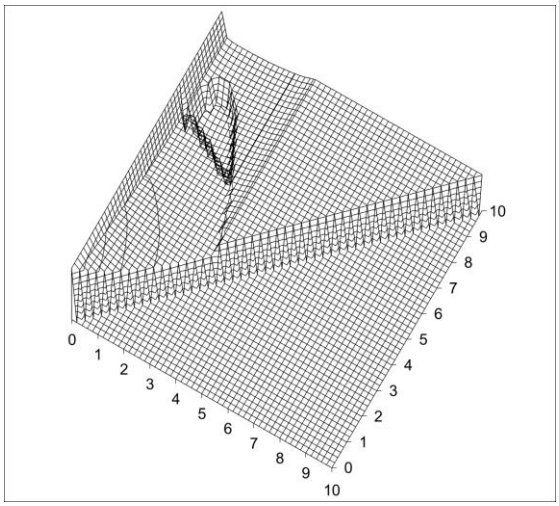
```
If Fchoice = 26 Then       'Angry Birds Space
  If x1 < 0 Or x1 > 10 Or x2 < 0 Or x2 > 10 Then
    constraint = "Fail"
    Exit Function
  End If
  abx1 = 0
```

```
   aby1 = 0
   abr1 = 1
   abGm1 = 1.25
   abx2 = 7
   aby2 = 4
   abr2 = 1
   abGm2 = 1
   abx3 = 2
   aby3 = 9
   abr3 = 0.5
   abGm1 = 0.25
   abcdr2m = 1
   abalpha2 = -3
   abalpha1 = -2
   abx = 1
   aby = 1
   abd = 0.06 + x1 * (0.2 - 0.06) / 10 '0.06 + (x1 - 3) / 50  'slingshot draw
   abangle = 36 * (-2 + x2 * 6 / 10) '(x2 - 4) * 36
   abv = 40 * abd ^ 2
   abvx = abv * Cos(2 * 3.14159 * abangle / 360)
   abvy = abv * Sin(2 * 3.14159 * abangle / 360)
   abn = 50 '5000
   abdt = 1
   f_of_x = 0
   Value1 = 300
   Value2 = 100
   Value3 = 300
   Value4 = 800
   Value5 = 100
   Value6 = 100
   value7 = 500
   For abtcount = 1 To 500
      For abncount = 1 To abn
         abx = abx + abvx * abdt / abn
         aby = aby + abvy * abdt / abn
         abrb1 = Sqr((abx - abx1) ^ 2 + (aby - aby1) ^ 2)
         abrb2 = Sqr((abx - abx2) ^ 2 + (aby - aby2) ^ 2)
         abrb3 = Sqr((abx - abx3) ^ 2 + (aby - aby3) ^ 2)
         If abrb1 <= abr1 Then Exit For  'crashed
         If abrb2 <= abr2 Then Exit For  'crashed
         If abrb3 <= abr3 Then Exit For  'crashed
         abax = abGm1 * (abx1 - abx) / abrb1 ^ 3 + abGm2 * (abx2 - abx) / abrb2 ^ 3 + abGm3 * (abx3 -
abx) / abrb3 ^ 3 - abcdr2m * abv * abvx * Exp(abalpha1 * (abrb1 - abr1)) - abcdr2m * abv * abvx *
Exp(abalpha2 * (abrb2 - abr2))
         abay = abGm1 * (aby1 - aby) / abrb1 ^ 3 + abGm2 * (aby2 - aby) / abrb2 ^ 3 + abGm3 * (aby3 -
aby) / abrb3 ^ 3 - abcdr2m * abv * abvy * Exp(abalpha1 * (abrb1 - abr1)) - abcdr2m * abv * abvy *
Exp(abalpha2 * (abrb2 - abr2))
         abvx = abvx + abdt * abax / abn
```

```
    abvy = abvy + abdt * abay / abn
    abv = Sqr(abvx ^ 2 + abvy ^ 2)
    abdragx = abcdr2m * abv * abvx * Exp(abalpha * (abrb2 - abr2)) / abax

    distance = Sqr((abx - 4) ^ 2 + (aby - 10) ^ 2)
    If Value1 > 0 Then
       increment = 3 * Exp(-2 * distance)
       f_of_x = f_of_x + increment
       Value1 = Value1 - increment
    End If
    distance = Sqr((abx - 7) ^ 2 + (aby - 8) ^ 2)
    If Value2 > 0 Then
       increment = 1 * Exp(-2 * distance)
       f_of_x = f_of_x + increment
       Value2 = Value2 - increment
    End If
    distance = Sqr((abx - 2.5) ^ 2 + (aby - 9) ^ 2)
    If Value3 > 0 Then
       increment = 3 * Exp(-2 * distance)
       f_of_x = f_of_x + increment
       Value3 = Value3 - increment
    End If
    distance = Sqr((abx - 7) ^ 2 + (aby - 5) ^ 2)
    If Value4 > 0 Then
       increment = 8 * Exp(-3 * distance)
       f_of_x = f_of_x + increment
       Value4 = Value4 - increment
    End If
    distance = Sqr((abx - 7) ^ 2 + (aby - 3) ^ 2)
    If Value5 > 0 Then
       increment = 1 * Exp(-2 * distance)
       f_of_x = f_of_x + increment
       Value5 = Value5 - increment
    End If
    distance = Sqr((abx - 6) ^ 2 + (aby - 4) ^ 2)
    If Value6 > 0 Then
       increment = 1 * Exp(-2 * distance)
       f_of_x = f_of_x + increment
       Value6 = Value6 - increment
    End If
    distance = Sqr((abx - 8) ^ 2 + (aby - 4) ^ 2)
    If value7 > 0 Then
       increment = 5 * Exp(-2 * distance)
       f_of_x = f_of_x + increment
       value7 = value7 - increment
    End If

  Next abncount
```

```
      If abrb1 <= abr1 Then Exit For  'crashed
      If abrb2 <= abr2 Then Exit For  'crashed
      If abrb3 <= abr3 Then Exit For  'crashed
      If abx < -10 Or abx > 30 Or aby < -10 Or aby > 30 Then Exit For
   Next abtcount
   f_of_x = -Sqr(f_of_x)
   f_of_x = 10 * (f_of_x + 30) / 30
   Exit Function
End If
```

Goddard Problem (#79) – There are many variations on this application named to honor Robert H. Goddard, liquid-fueled rocket scientist. The objective is to schedule the thrust to minimize fuel consumption to achieve a desired rocket height. If the initial thrust is not high enough, then the rocket never lifts off the launch pad and burns all the fuel. The best initial thrust is 100%, to get the rocket moving. But, once moving upward, the faster it goes, the greater is the air drag, and continuing full thrust leads to excessive velocity and excessive drag, which wastes fuel. In this simple exercise, the initial thrust is 100%, and the objective is to find the elevation to cut the thrust to 20% then the next elevation to cut it to 0%. Even at 0% thrust, the upward velocity of the rocket continues until gravitational pull and air drag counter the momentum.



Thanks to Thomas Hays for pointing me to this in the fall of 2013. The model and coefficient values are from http://bocop.saclay.inria.fr/?page_id=465.

The OF has flat spots, cliff discontinuities, and hard constraints. There is small sensitivity to the second stage of thrust, and numerical time discretization of the dynamic model has a visible impact on the OF surface.

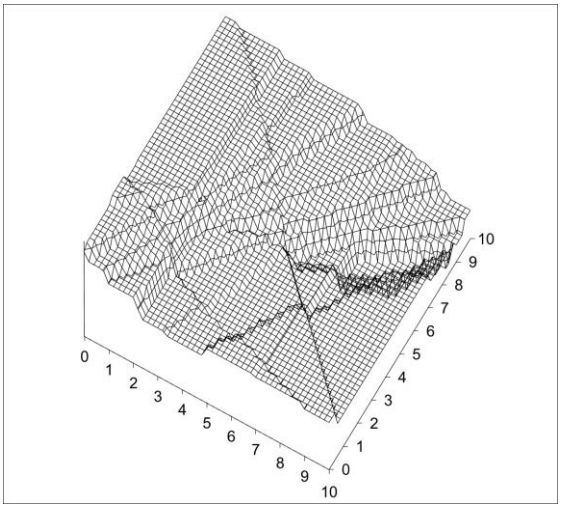
The VBA Code is:

```
   If Fchoice = 79 Then    'Goddard Problem 3.  Start with u1 thrust = 1, then
      'find elevation to switch to u2=0.2 and then elevation to switch fuel off
      'to maximize remaining fuel content to reach desired height
      'time is i79*dtprime*7915, sec
      'elevation is (rprime-1)*20170, thousands of feet (rocket can get to 243k feet, 46 miles)

      If x1 > 10 Or x2 > 10 Or x1 < 0 Or x2 < 0 Then
         constraint = "FAIL"
         Exit Function
      End If
      rprimeend = 1.005       'about 100k feet
      rprime1 = 1 + (rprimeend - 1) * x1 / 10
```

```
    rprime2 = 1 + (rprimeend - 1) * x2 / 10
    If rprime1 > rprime2 Then
      constraint = "FAIL"
      Exit Function
    End If

    Stage79 = 1
    mprime0 = 1          'initial scaled mass of rocket with fuel
    rprime0 = 1          'initially on the Earth's surface, scaled radius
    vprime0 = 0          'initial scaled velocity
    dtprime = 0.0001     'small dt is required to make numerical discretization inconsequential, should
use 0.00001
    For i79 = 1 To 100000    'takes about a quarter this many iterations for a simulation
    If Stage79 = 1 Then influence = 1
    If Stage79 = 2 Then influence = 0.2
    If Stage79 = 3 Then influence = 0
      mprime = mprime0 - dtprime * 24.5 * influence
      vprime = vprime0 + dtprime * (3.5 * influence / mprime0 - 1 / rprime0 ^ 2 - 310 * vprime0 ^ 2 *
Exp(-500 * (rprime0 - 1)) / mprime0)
      rprime = rprime0 + dtprime * vprime0
      If vprime < 0 Then Exit For    'falling
      If mprime < 0.2 Then Exit For   'fuel out not a good trial
      If rprime > rprime1 And Stage79 = 1 Then  'entered stage 2
        dtincrement = dtprime * (rprime - rprime1) / (rprime - rprime0) 'recalculate last dt
        mprime = mprime0 - dtincrement * 24.5 * influence
        vprime = vprime0 + dtincrement * (3.5 * influence / mprime0 - 1 / rprime0 ^ 2 - 310 * vprime0
^ 2 * Exp(-500 * (rprime0 - 1)) / mprime0)
        rprime = rprime0 + dtincrement * vprime0
        Stage79 = 2
      End If
      If rprime > rprime2 And Stage79 = 2 Then  'entered stage 3
        dtincrement = dtprime * (rprime - rprime2) / (rprime - rprime0) 'recalculate last dt
        mprime = mprime0 - dtincrement * 24.5 * influence
        vprime = vprime0 + dtincrement * (3.5 * influence / mprime0 - 1 / rprime0 ^ 2 - 310 * vprime0
^ 2 * Exp(-500 * (rprime0 - 1)) / mprime0)
        rprime = rprime0 + dtincrement * vprime0
        Stage79 = 3
      End If
      mprime0 = mprime
      vprime0 = vprime
      rprime0 = rprime
      If rprime > rprimeend Then Exit For  'made it 1.005 is about 100k ft.
    Next i79
    f_of_x = -mprime '+ 10 * vprime ^ 2  'plus penalty for excessive velocity (unnecessary)
'     If rprime < rprimeend Then f_of_x = f_of_x + (1000 * (rprimeend - rprime)) ^ 2        'soft penalty for
not making height
'     f_of_x = 10 * (f_of_x + 0.353) / 24
```

```
    If rprime < rprimeend Then f_of_x = (100 * (rprimeend - rprime)) ^ 2  'change OF penalty for not meeting height
    f_of_x = 10 * (f_of_x + 0.25826) / 0.50824
    Exit Function
  End If
```

Rank Model (#84) – The objective is to create a formula which can determine competitiveness of players from their attributes. Perhaps you need a method to invite people to be junior members of your school team, and want to know who has the best promise to develop into a winner at the varsity level. Perhaps you want to know whether your 6-year old child has talent, and whether you should invest lots of time and money in developing them for a possible place on the Olympic team. Perhaps you want to know if you should join the tennis ladder at work.



The hope is that there are fundamental physical attributes of the individuals that can indicate their competitive fitness. For instance, reflex rate could be easily measured in a laboratory test of the individual's delay in hand response to visual signals, and such reflex rate seems to be a relevant metric to imply fast responses on the tennis court, which seems important to winning. Smaller would be better. Another measureable attribute might be how far a person can reach when standing. Larger would be better. There are several such attributes, and the hope is that a model of overall fitness can be developed from the following power-law function: $F = {f_1}^a {f_2}^b {f_3}^c$ …, where F is the overall player fitness, f_i are the lab-measured attributes, and coefficients a, b, c, etc. are adjustable model parameters (these are the DVs in the optimization).

This model indicates that if one factor is very low (perhaps $f_2 = 0.02$), even if two are very high (perhaps $f_1 = 0.93$ and $f_3 = 0.89$), the player is not as fit for the game as one with mediocre values for all attributes (perhaps $f_1 = f_2 = f_3 = 0.7$).

The model will be developed by taking N number of high-level players, testing them for their attribute values, and have them play in tournaments under various conditions (indoor, outdoor, clay court, morning, evening, etc.). The model will be adjusted so that the rank predicted by the overall fitness model best matches the average rank found from the tournaments.

Typical of rank models, there are cliff discontinuities and flat spots. If overall fitness values for players A, B, and C are 10, 7 and 2, then their ranks are 1st, 2nd, and 3rd. If the fitness values were 8, 7.1 and 2, the ranking does not change. If 7.3, 7.29, and 2, still not changed. However with a tiny change to 7.3, 7.301, and 2, the ranking abruptly changes to 2nd, 1st, and 3rd.
In this exercise, I choose to use three metrics, a geometric mean for the overall fitness, and to set the power of the first factor to unity. As long as a>0 the $F = {f_1}^a {f_2}^b {f_3}^c$ functionality can be raised to the 1/a power without changing ranking. Or equivalently, set a=1. Further, the b- and c-values are fixed and further exponentiation does not change the ranking.

$$F = \sqrt[1+b+c]{f_1 {f_2}^b {f_3}^c}$$

This is a 2-DV application. The example uses 3 contestants, with 3 attributes each, and seeks to best match the ranking from 6 contests between the three players.

The VBA code is:

```
  If Fchoice = 84 Then    'rank exploration - 6 sets of 3 contestants, 3 attributes each - what is model that best matches ranks
    x11 = -2 + x1 / 2   'model parameter 1
    x22 = -2 + x2 / 2   'model parameter 2
    If x1 < 0 Or x1 > 10 Or x2 < 0 Or x2 > 10 Then      'prevent extreme values
      constraint = "FAIL"
      Exit Function
    End If
                '   Set 1
    FA184 = 0.4       'Player attributes - Player A, Attribute 1
    FA284 = 0.3
    FA384 = 0.9
    FB184 = 0.4
    FB284 = 0.6
    FB384 = 0.5
    FC184 = 0.7
    FC284 = 0.7
    FC384 = 0.8
    fA84 = (FA184 * FA284 ^ x11 * FA384 ^ x22) '^ (1 / (x11 + x22 + 1))  'composite fitness
    fB84 = (FB184 * FB284 ^ x11 * FB384 ^ x22) '^ (1 / (x11 + x22 + 1))
    fC84 = (FC184 * FC284 ^ x11 * FC384 ^ x22) '^ (1 / (x11 + x22 + 1))
    minf84 = fA84                   ' find worst
    If fB84 < minf84 Then minf84 = fB84
    If fC84 < minf84 Then minf84 = fC84
    maxf84 = fA84                   ' find best
    If fB84 > maxf84 Then maxf84 = fB84
    If fC84 > maxf84 Then maxf84 = fC84
    RankA = 2                       ' Assign rank - very primitive approach
    RankB = 2
    RankC = 2
    If fA84 = maxf84 Then RankA = 1
    If fB84 = maxf84 Then RankB = 1
    If fC84 = maxf84 Then RankC = 1
    If fA84 = minf84 Then RankA = 3
    If fB84 = minf84 Then RankB = 3
    If fC84 = minf84 Then RankC = 3
    f_of_x = ((RankC - 1) ^ 2) / 1 + ((RankB - 2) ^ 2) / 2 + ((RankA - 3) ^ 2) / 3  'Assess - compare model to actual
                '   Set 2
    FA184 = 0.3       'Player attributes
    FA284 = 0.5
```

```
FA384 = 0.5
FB184 = 0.7
FB284 = 0.8
FB384 = 0.2
FC184 = 0.5
FC284 = 0.5
FC384 = 0.6
fA84 = (FA184 * FA284 ^ x11 * FA384 ^ x22) '^ (1 / (x11 + x22 + 1))  'composite
fB84 = (FB184 * FB284 ^ x11 * FB384 ^ x22) '^ (1 / (x11 + x22 + 1))
fC84 = (FC184 * FC284 ^ x11 * FC384 ^ x22) '^ (1 / (x11 + x22 + 1))
minf84 = fA84                   ' find worst
If fB84 < minf84 Then minf84 = fB84
If fC84 < minf84 Then minf84 = fC84
maxf84 = fA84                   ' find best
If fB84 > maxf84 Then maxf84 = fB84
If fC84 > maxf84 Then maxf84 = fC84
RankA = 2                       ' Assign rank
RankB = 2
RankC = 2
If fA84 = maxf84 Then RankA = 1
If fB84 = maxf84 Then RankB = 1
If fC84 = maxf84 Then RankC = 1
If fA84 = minf84 Then RankA = 3
If fB84 = minf84 Then RankB = 3
If fC84 = minf84 Then RankC = 3
f_of_x = f_of_x + ((RankC - 1) ^ 2) / 1 + ((RankB - 2) ^ 2) / 2 + ((RankA - 3) ^ 2) / 3 'Assess
                '   Set 3
FA184 = 0.6       'Player attributes
FA284 = 0.5
FA384 = 0.6
FB184 = 0.7
FB284 = 0.6
FB384 = 0.5
FC184 = 0.8
FC284 = 0.7
FC384 = 0.5
fA84 = (FA184 * FA284 ^ x11 * FA384 ^ x22) '^ (1 / (x11 + x22 + 1))  'composite
fB84 = (FB184 * FB284 ^ x11 * FB384 ^ x22) '^ (1 / (x11 + x22 + 1))
fC84 = (FC184 * FC284 ^ x11 * FC384 ^ x22) '^ (1 / (x11 + x22 + 1))
minf84 = fA84                   ' find worst
If fB84 < minf84 Then minf84 = fB84
If fC84 < minf84 Then minf84 = fC84
maxf84 = fA84                   ' find best
If fB84 > maxf84 Then maxf84 = fB84
If fC84 > maxf84 Then maxf84 = fC84
RankA = 2                       ' Assign rank
RankB = 2
RankC = 2
```

```
If fA84 = maxf84 Then RankA = 1
If fB84 = maxf84 Then RankB = 1
If fC84 = maxf84 Then RankC = 1
If fA84 = minf84 Then RankA = 3
If fB84 = minf84 Then RankB = 3
If fC84 = minf84 Then RankC = 3
f_of_x = f_of_x + ((RankC - 1) ^ 2) / 1 + ((RankB - 2) ^ 2) / 2 + ((RankA - 3) ^ 2) / 3 'Assess
                        '   Set 4
FA184 = 0.6        'Player attributes
FA284 = 0.5
FA384 = 0.5
FB184 = 0.7
FB284 = 0.6
FB384 = 0.5
FC184 = 0.8
FC284 = 0.8
FC384 = 0.4
fA84 = (FA184 * FA284 ^ x11 * FA384 ^ x22) '^ (1 / (x11 + x22 + 1))  'composite
fB84 = (FB184 * FB284 ^ x11 * FB384 ^ x22) '^ (1 / (x11 + x22 + 1))
fC84 = (FC184 * FC284 ^ x11 * FC384 ^ x22) '^ (1 / (x11 + x22 + 1))
minf84 = fA84                   ' find worst
If fB84 < minf84 Then minf84 = fB84
If fC84 < minf84 Then minf84 = fC84
maxf84 = fA84                   ' find best
If fB84 > maxf84 Then maxf84 = fB84
If fC84 > maxf84 Then maxf84 = fC84
RankA = 2                       ' Assign rank
RankB = 2
RankC = 2
If fA84 = maxf84 Then RankA = 1
If fB84 = maxf84 Then RankB = 1
If fC84 = maxf84 Then RankC = 1
If fA84 = minf84 Then RankA = 3
If fB84 = minf84 Then RankB = 3
If fC84 = minf84 Then RankC = 3
f_of_x = f_of_x + ((RankC - 1) ^ 2) / 1 + ((RankB - 2) ^ 2) / 2 + ((RankA - 3) ^ 2) / 3 'Assess
                        '   Set 5
FA184 = 0.1        'Player attributes
FA284 = 0.6
FA384 = 0.2
FB184 = 0.5
FB284 = 0.2
FB384 = 0.2
FC184 = 0.3
FC284 = 0.3
FC384 = 0.3
fA84 = (FA184 * FA284 ^ x11 * FA384 ^ x22) '^ (1 / (x11 + x22 + 1))  'composite
fB84 = (FB184 * FB284 ^ x11 * FB384 ^ x22) '^ (1 / (x11 + x22 + 1))
```

```
fC84 = (FC184 * FC284 ^ x11 * FC384 ^ x22) '^ (1 / (x11 + x22 + 1))
minf84 = fA84                   ' find worst
If fB84 < minf84 Then minf84 = fB84
If fC84 < minf84 Then minf84 = fC84
maxf84 = fA84                   ' find best
If fB84 > maxf84 Then maxf84 = fB84
If fC84 > maxf84 Then maxf84 = fC84
RankA = 2                       ' Assign rank
RankB = 2
RankC = 2
If fA84 = maxf84 Then RankA = 1
If fB84 = maxf84 Then RankB = 1
If fC84 = maxf84 Then RankC = 1
If fA84 = minf84 Then RankA = 3
If fB84 = minf84 Then RankB = 3
If fC84 = minf84 Then RankC = 3
f_of_x = f_of_x + ((RankC - 1) ^ 2) / 1 + ((RankA - 2) ^ 2) / 2 + ((RankB - 3) ^ 2) / 3 'Assess
                        '   Set 6
FA184 = 0.7       'Player attributes
FA284 = 0.4
FA384 = 0.8
FB184 = 0.8
FB284 = 0.3
FB384 = 0.9
FC184 = 0.8
FC284 = 0.5
FC384 = 0.6
fA84 = (FA184 * FA284 ^ x11 * FA384 ^ x22) '^ (1 / (x11 + x22 + 1))  'composite
fB84 = (FB184 * FB284 ^ x11 * FB384 ^ x22) '^ (1 / (x11 + x22 + 1))
fC84 = (FC184 * FC284 ^ x11 * FC384 ^ x22) '^ (1 / (x11 + x22 + 1))
minf84 = fA84                   ' find worst
If fB84 < minf84 Then minf84 = fB84
If fC84 < minf84 Then minf84 = fC84
maxf84 = fA84                   ' find best
If fB84 > maxf84 Then maxf84 = fB84
If fC84 > maxf84 Then maxf84 = fC84
RankA = 2                       ' Assign rank
RankB = 2
RankC = 2
If fA84 = maxf84 Then RankA = 1
If fB84 = maxf84 Then RankB = 1
If fC84 = maxf84 Then RankC = 1
If fA84 = minf84 Then RankA = 3
If fB84 = minf84 Then RankB = 3
If fC84 = minf84 Then RankC = 3
f_of_x = f_of_x + ((RankC - 1) ^ 2) / 1 + ((RankA - 2) ^ 2) / 2 + ((RankB - 3) ^ 2) / 3 'Assess
f_of_x = Sqr(f_of_x)              ' Transformed to visualize contours
f_of_x = 10 * (f_of_x - 0) / 5.066
```

```
    Exit Function
  End If
```

Reported on ResearchGate summer 2015 as sources of two dimensional optimization challenge problems:
http://www.mat.univie.ac.at/~neum/glopt/test.html
http://www.gamsworld.org/performance/selconglobal/selcongloballib.htm
http://www-optima.amp.i.kyoto-.ac.jp/member/student/hedar/Hedar_files/TestGO_files/Page364.htm
http://www.geatbx.com/docu/fcnindex-01.html
http://coco.gforge.inria.fr/doku.php?id=bbob-2013-downloads
https://en.wikipedia.org/wiki/Test_functions_for_optimization
Http://www.mat.univie.ac.at/~vpk/math/funcs.html
http://www.cs.bham.ac.uk/~xin/papers/published_tec_jul99.pdf
http://www.zsd.ict.pwr.wroc.pl/files/docs/functions.pdf
http://www.ntu.edu.sg/home/EPNSugan/index_files/CEC2013/CEC2013.htm
https://en.wikipedia.org/wiki/Test_functions_for_optimization
https://www.researchgate.net/profile/Sergey_Porotsky/publications