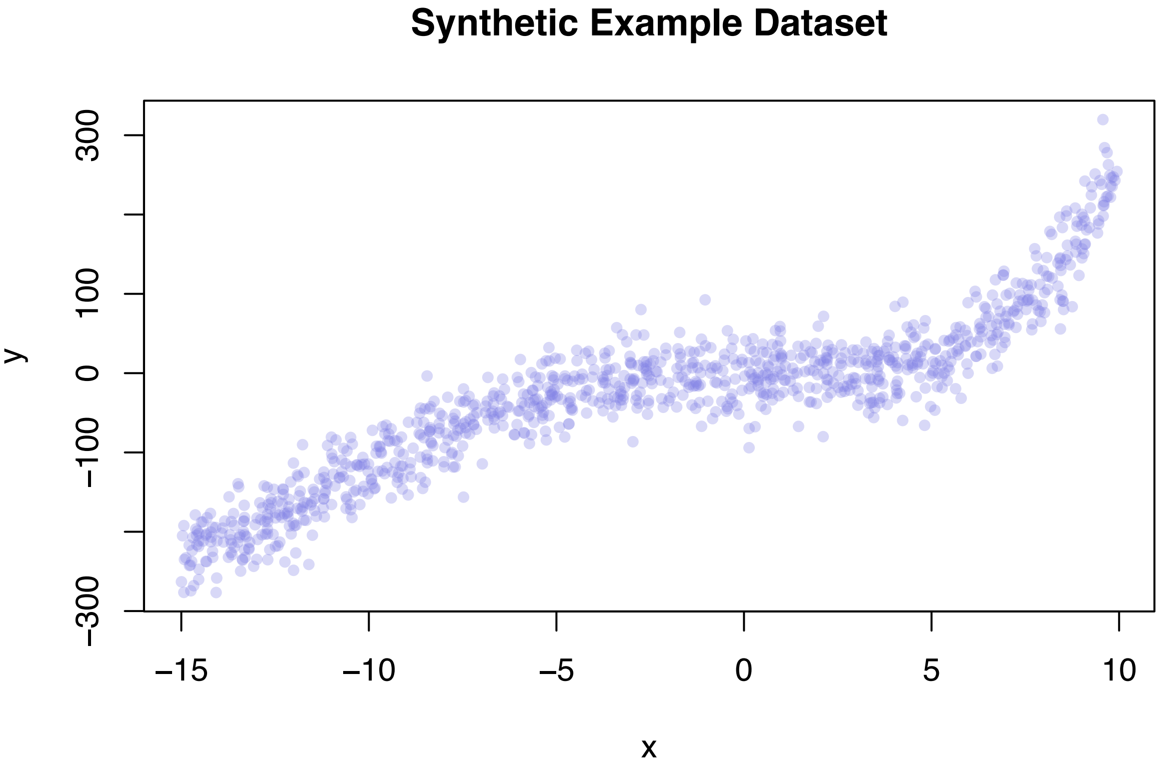
Tutorial 2: Finding an Optimal Equation by Navigating Through a Term Space

Problem Statement

Consider again the synthetic data that was created in **Tutorial 1**. Suppose that we were only provided with the data and, unlike in **Tutorial 1**, had no knowledge of the best terms to be included in a functional representation of said data. In this example, we shall use Optimus to determine which terms should be used in a least squares fitting of the data to achieve a representation with low RMSD while avoiding overfitting the data with too complicated equation.

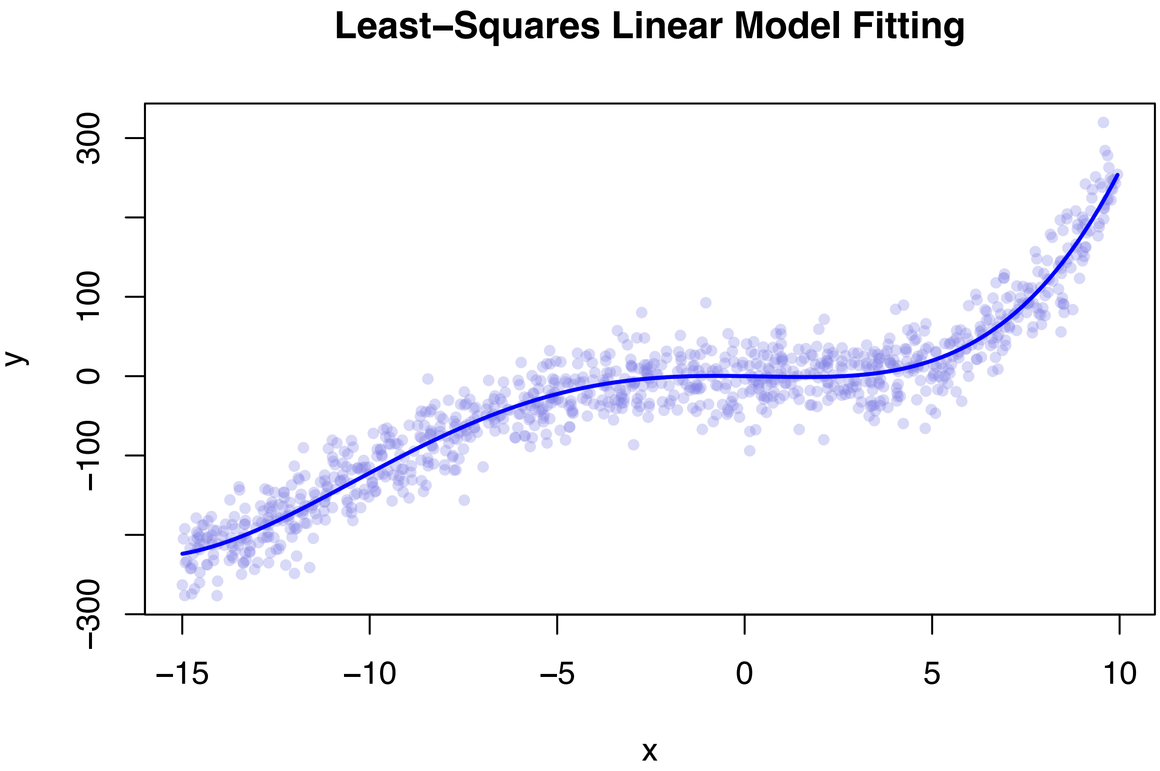
Let us start by generating the same data which was used in **Tutorial 1**:

```
set.seed(845)
x <- runif(1000, min=-15, max=10)
y <- -1.0*x - 0.3*x^2 + 0.2*x^3 + 0.01*x^4 + rnorm(length(x), mean=0, sd=30)
```



From **Tutorial 1**, we know that if presented with this data and under the assumption that the most appropriate model to describe the data is $k_1x + k_2x^2 + k_3x^3 + k_4x^4$, the Least Squares model fitting is $y = -0.741x - 0.307x^2 + 0.198x^3 + 0.010x^4$. We also know that the RMSD between the observed data y and the linear model fitting outcome is:

```
## [1] 28.82655
```



Defining Optimus Inputs

Let us first define an ordered set *terms* that is a collection of candidate terms to include in the representation of the data:

$terms = \{x,\, x^2,\, x^3,\, x^4,\, x^5,\, x^6,\, x^7,\, x^8,\, x^9,\, x^{10},\, e^x,\, |x|,\, sin(x),\, cos(x),\, tan(x),\, sin(x)cos(x),\, sin^2(x),\, cos^2(x),\, sin(x^2), sin(x^3), cos(x^2), cos(x^3), sin(x^3)cos(-x), cos(x^3)sin(-x), sin(x^5)cos(-x), cos(x^5)sin(-x), e^x sin(x),\, e^x cos(x),\, |x|sin(x),\, |x|cos(x)\}$

Let $terms_i$ denote the i^{th} term in the set *terms* (for example, $terms_{14} = cos(x)$). We shall use the model:

$$y = b + \sum_{i=1}^{\mathbf{card}(terms)} k_i c_i terms_i$$

where each k_i is a binary variable (meaning a variable taking a value of either 0 or 1) indicating whether the i^{th} term is included in the representation, each c_i is a non-zero coefficient for the i^{th} term and b is a real number (the intercept). In our case, $\mathbf{card}(terms) = 30$ so explicitly, our model is:

$y = b + k_1c_1x + k_2c_2x^2 + k_3c_3x^3 + k_4c_4x^4 + k_5c_5x^5 + k_6c_6x^6 + k_7c_7x^7 + k_8c_8x^8 + k_9c_9x^9 + k_{10}c_{10}x^{10} + k_{11}c_{11}e^x + k_{12}c_{12}|x| + k_{13}c_{13}six(x) + k_{14}c_{14}cos(x) + k_{15}c_{15}tan(x) + k_{16}c_{16}sin(x)cos(x) + k_{17}c_{17}sin^2(x) + k_{18}c_{18}cos^2(x) + k_{19}c_{19}sin(x^2) + k_{20}c_{20}sin(x^3) + k_{21}c_{21}cos(x^2) + k_{22}c_{22}cos(x^3) + k_{23}c_{23}sin(x^3)cos(-x) + k_{24}c_{24}cos(x^3)sin(-x) + k_{25}c_{25}sin(x^5)cos(-x) + k_{26}c_{26}cos(x^5)sin(-x) + k_{27}c_{27}e^x sin(x) + k_{28}c_{28}e^x cos(x) + k_{29}c_{29}|x|sin(x) + k_{30}c_{30}|x|cos(x)$

Formally, `K` will be a numeric vector of length $\mathbf{card}(terms)$ whose i^{th} entry is k_i. `K` uniquely specifies a set $activeTerms = \{terms_i \;\forall\, i \,|k_i = 1\}$. Note that $activeTerms \subseteq terms$. Each binary variable k_i should be initialized randomly as below:

```
K <- c(term1=rbinom(n=1, size=1, prob=0.5),
       term2=rbinom(n=1, size=1, prob=0.5),
       term3=rbinom(n=1, size=1, prob=0.5),
       term4=rbinom(n=1, size=1, prob=0.5),
       term5=rbinom(n=1, size=1, prob=0.5),
       term6=rbinom(n=1, size=1, prob=0.5),
       term7=rbinom(n=1, size=1, prob=0.5),
       term8=rbinom(n=1, size=1, prob=0.5),
       term9=rbinom(n=1, size=1, prob=0.5),
       term10=rbinom(n=1, size=1, prob=0.5),
       term11=rbinom(n=1, size=1, prob=0.5),
       term12=rbinom(n=1, size=1, prob=0.5),
       term13=rbinom(n=1, size=1, prob=0.5),
       term14=rbinom(n=1, size=1, prob=0.5),
       term15=rbinom(n=1, size=1, prob=0.5),
       term16=rbinom(n=1, size=1, prob=0.5),
       term17=rbinom(n=1, size=1, prob=0.5),
       term18=rbinom(n=1, size=1, prob=0.5),
       term19=rbinom(n=1, size=1, prob=0.5),
       term20=rbinom(n=1, size=1, prob=0.5),
       term21=rbinom(n=1, size=1, prob=0.5),
       term22=rbinom(n=1, size=1, prob=0.5),
       term23=rbinom(n=1, size=1, prob=0.5),
       term24=rbinom(n=1, size=1, prob=0.5),
       term25=rbinom(n=1, size=1, prob=0.5),
       term26=rbinom(n=1, size=1, prob=0.5),
       term27=rbinom(n=1, size=1, prob=0.5),
       term28=rbinom(n=1, size=1, prob=0.5),
       term29=rbinom(n=1, size=1, prob=0.5),
       term30=rbinom(n=1, size=1, prob=0.5))
```

Next, we must define the model function `m()` that will operate on the parameter snapshot `K` and return an observable object `O`. For a given set *activeTerms* specified by `K`, `m()` will fit a linear model to the data using the entries in *activeTerms* and using the built in generalised linear model (`glm()`) function in R, thereby determining values for the variables c_i and b. Accordingly, `m()` will require access to the variables x and y, which will be provided as entries in `DATA`, a variable of type list, as in **Tutorial 1**. The object `O` will be the corresponding output of the function `glm()`. In the case that the set *activeTerms* is the empty set (meaning that all entries in `K` are 0), `m()` will fit a model using the relationship $y \sim x$.

```
DATA   <- NULL
DATA$x <- x
DATA$y <- y

m <- function(K, DATA){
  y <- DATA$y
  x <- DATA$x

  terms <- c("+x",
             "+I(x^2)",
             "+I(x^3)",
             "+I(x^4)",
             "+I(x^5)",
             "+I(x^6)",
             "+I(x^7)",
             "+I(x^8)",
             "+I(x^9)",
             "+I(x^10)",
             "+I(exp(x))",
             "+I(abs(x))",
             "+I(sin(x))",
             "+I(cos(x))",
             "+I(tan(x))",
             "+I(sin(x)*cos(x))",
             "+I((sin(x))^2)",
             "+I((cos(x))^2)",
             "+I(sin(x^2))",
             "+I(sin(x^3))",
             "+I(cos(x^2))",
             "+I(cos(x^3))",
             "+I(sin(x^3)*cos(-x))",
             "+I(cos(x^3)*sin(-x))",
             "+I(sin(x^5)*cos(-x))",
             "+I(cos(x^5)*sin(-x))",
             "+I(exp(x)*sin(x))",
             "+I(exp(x)*cos(x))",
             "+I(abs(x)*sin(x))",
             "+I(abs(x)*cos(x))")

  ind.terms <- which(K == 1)
  if(length(ind.terms)!=0){
    equation <- paste(c("y~",terms[ind.terms]), collapse="")
  } else {
    equation <-"y~x" # In case there are no active terms, use a simple linear model.
  }

  O <- glm(equation, data=environment())

  return(O)
}
```

Having defined `m`, we can now proceed to define the function `u`, which will determine how well a given configuration of parameters `K` is performing by operating on the observable object `O` outputted by `m()` and on the variable `DATA`. Here, to quantify (and thus be able to compare) the desirability of a given model for the data, we will employ the Akaike Information Criterion (AIC) from information theory, defined as follows:

$$AIC(M) = 2p - 2ln(L)$$

where p is the number of parameters in the fitted model, L is the maximum likelihood of the model M given the data.

The target representation will be the fitted model M (whose terms are elements of $terms$) that minimises the AIC. It is important to note that the $2p$ term in the AIC penalises overfitting by increasing AIC as function of the number of parameters, while the $-2ln(L)$ term rewards models that better represent the data by decreasing AIC as a function of the likelihood of the model.

As articulated in **Tutorial 1**, the output of `u()` should have a component `E` holding a pseudo energy for the parameter snapshot `K`, and a component `Q` that can be used for plotting the optimisation process. In this case, `E` will be equal to the value of AIC (implemented using the built in `AIC()` function in R) and `Q` will be equal to the RMSD between the predicted values of y from the fitted model and the actual y values, used for plotting purpose. Consequently, `u()` will need access to the variable y. The definition of `u()` is below:

```
u <- function(O, DATA){
  y <- DATA$y

  Q <- sqrt(mean((O$fitted.values-y)^2))
  E <- AIC(O)/1000 # Akaike's information criterion.

  result   <- NULL
  result$Q <- Q
  result$E <- E
  return(result)
}
```

Finally, we need to define the rule function `r()`. We will adopt the following simple procedure: randomly select an equation entry from `K` and switch its value to the other binary value (on to off, or off to on).

```
r <- function(K){
  K.new <- K
  # Randomly selecting a term:
  K.ind.toalter <- sample(size=1, x=1:length(K.new))
  # If the term is on (1), switching it off (0) or vice versa:
  if(K.new[K.ind.toalter]==1){
    K.new[K.ind.toalter] <- 0
  } else {
    K.new[K.ind.toalter] <- 1
  }
  return(K.new)
}
```

Having defined all the necessary inputs, we are now ready to call Optimus.

An important remark is that modelling this problem in this manner results in an objective function (AIC) that is not smooth because small changes in the parameter set `K` (as defined by `r()`) can produce significantly large changes in the objective value. The equation, i.e. the system itself, changes from one step to another. Therefore, an entirely different model is being used to fit the data at each step. Optimisation procedures in such cases are in danger to be trapped in certain minima due to a major change in pseudo temperatures necessary to overcome barriers while the system abruptly evolves. Despite this, we will see that Optimus will get the job done by arriving at good solutions largely as a consequence of its adaptive thermoregulation and acceptance-ratio-guided optimisation procedure.

Acceptance Ratio Simulated Annealing Optimus Run

In addition to the inputs defined above, Optimus can optionally take other inputs to dictate the optimisation process (see the Advanced User Manual), all of which have built in default values and some of which will be altered in this example due to the increased computational complexity of the model defined in this tutorial compared to that of **Tutorial 1**. The variable `NUMITER` represents the number of iterations of the optimisation process (per core) and has a default value of 1 000 000. For this example, 200 000 iterations will be used to reduce the running time of Optimus given that each iteration is more computationally demanding than in **Tutorial 1**. The variable `CYCLES` (unique to Acceptance Ratio Annealing Optimus runs) denotes the number of acceptance ratio annealing cycles. Its default value is 10, however it will be set to 2 in this example so that each annealing cycle has 100 000 iterations just as in **Tutorial 1** (the number of steps per cycle is calculated as `NUMITER/CYCLES`). Lastly, the variable `DUMP.FREQ`, the frequency (in steps) with which the best found model is assessed and outputted by the function, will be set to 100 000 (its default value is 10 000).

Let us again investigate the Simulated Annealing (SA) version of Optimus on 4 processors, which can be executed as follows:

```
Optimus(NCPU=4,
        OPT.TYPE="SA", OPTNAME="term_4_SA",
        NUMITER=200000, CYCLES=2, DUMP.FREQ=100000, LONG=FALSE,
        K.INITIAL=K, rDEF=r, mDEF=m, uDEF=u, DATA=DATA)
```

Interestingly, each of the 4 computing cores arrive at the same solution in this instance.

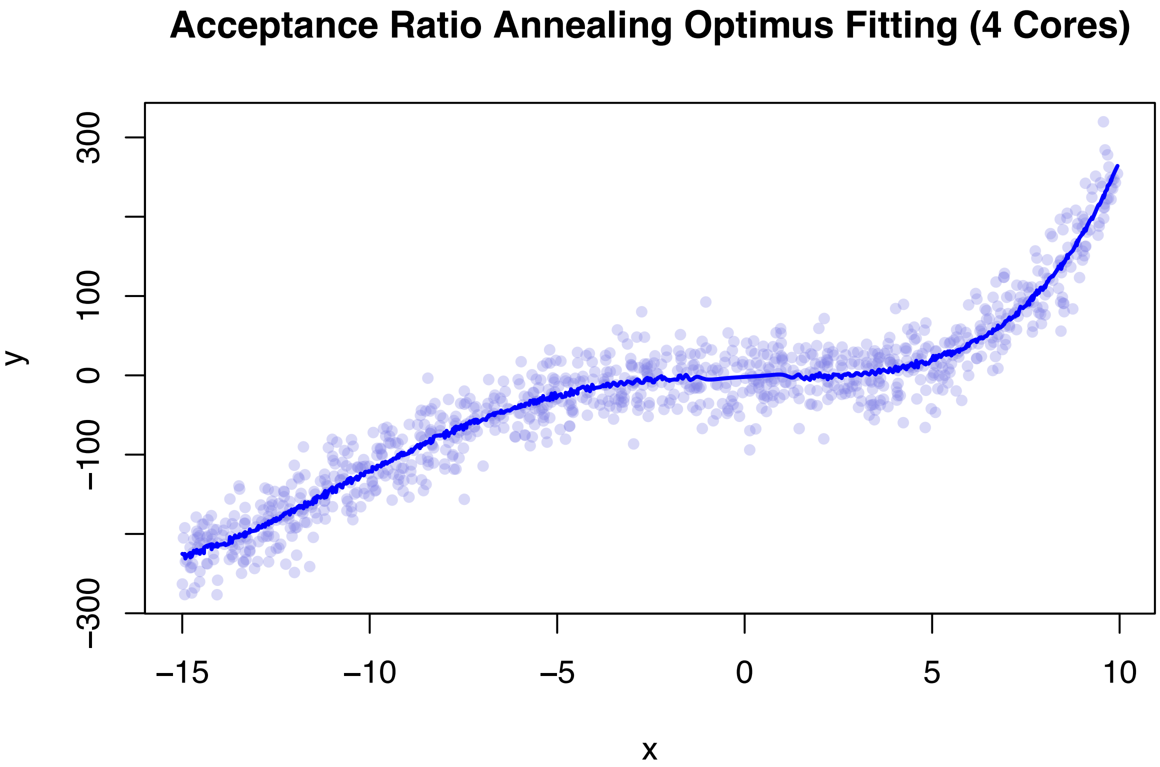


Table 4: 4-core Acceptance Ratio Simulated Annealing results from Optimus.

	CPU 1	CPU 2	CPU 3	CPU 4
E (AIC)	9.567	9.567	9.567	9.567
Q (RMSD)	28.693	28.693	28.693	28.693
Term 1	0.000	0.000	0.000	0.000
Term 2	1.000	1.000	1.000	1.000
Term 3	1.000	1.000	1.000	1.000
Term 4	1.000	1.000	1.000	1.000
Term 5	0.000	0.000	0.000	0.000
Term 6	0.000	0.000	0.000	0.000
Term 7	0.000	0.000	0.000	0.000
Term 8	0.000	0.000	0.000	0.000
Term 9	0.000	0.000	0.000	0.000
Term 10	0.000	0.000	0.000	0.000
Term 11	1.000	1.000	1.000	1.000
Term 12	0.000	0.000	0.000	0.000
Term 13	0.000	0.000	0.000	0.000
Term 14	0.000	0.000	0.000	0.000
Term 15	0.000	0.000	0.000	0.000
Term 16	0.000	0.000	0.000	0.000

	CPU 1	CPU 2	CPU 3	CPU 4
Term 17	0.000	0.000	0.000	0.000
Term 18	0.000	0.000	0.000	0.000
Term 19	0.000	0.000	0.000	0.000
Term 20	1.000	1.000	1.000	1.000
Term 21	0.000	0.000	0.000	0.000
Term 22	0.000	0.000	0.000	0.000
Term 23	0.000	0.000	0.000	0.000
Term 24	0.000	0.000	0.000	0.000
Term 25	0.000	0.000	0.000	0.000
Term 26	1.000	1.000	1.000	1.000
Term 27	0.000	0.000	0.000	0.000
Term 28	0.000	0.000	0.000	0.000
Term 29	0.000	0.000	0.000	0.000
Term 30	0.000	0.000	0.000	0.000

Thus, the optimal functional representation found by Optimus has the following form:

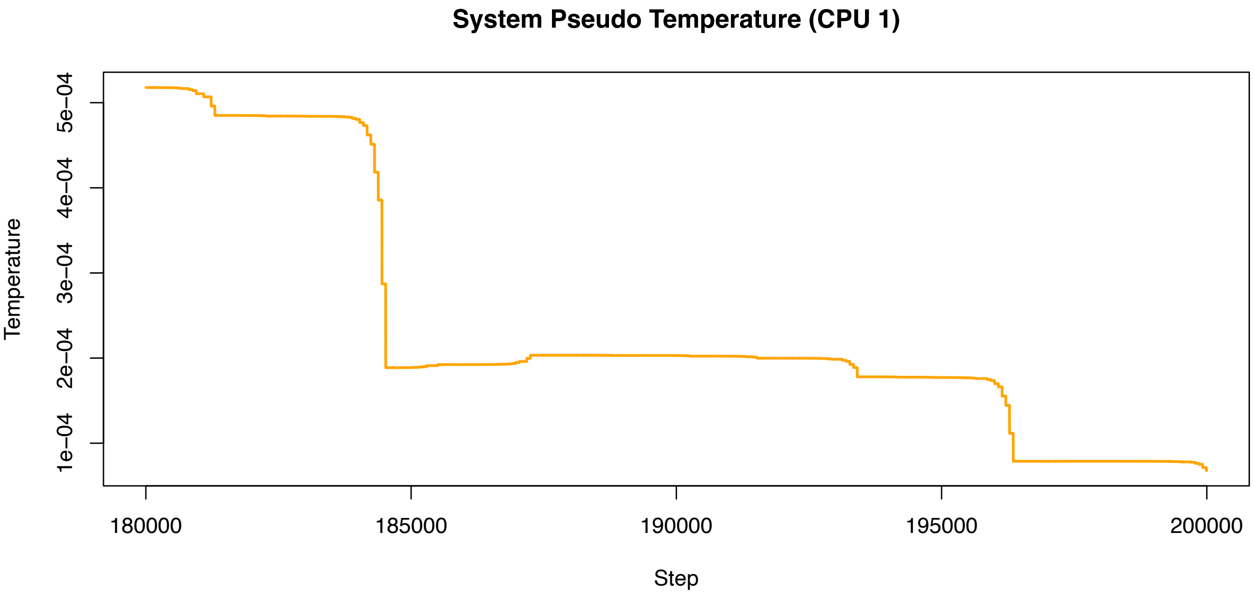
$$y = b + c_2x^2 + c_3x^3 + c_4x^4 + c_{11}e^x + c_{20}sin(x^3) + c_{26}cos(x^5)sin(-x)$$

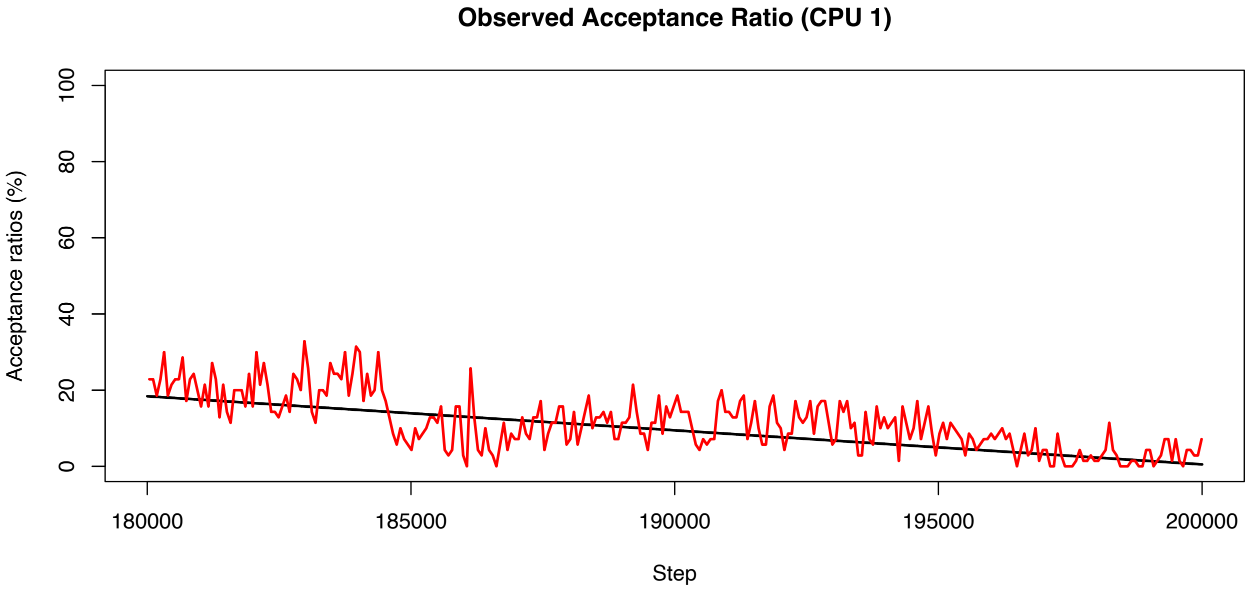
Below is the explicit representation after determining the coefficients c_i and b:

```
##
## Call:  glm(formula = equation, data = environment())
##
## Coefficients:
##           (Intercept)                I(x^2)                I(x^3)
##             -1.988699             -0.253437              0.178807
##                I(x^4)             I(exp(x))          I(sin(x^3))
##              0.008567              0.001569              1.878796
## I(cos(x^5) * sin(-x))
##             -2.770457
##
## Degrees of Freedom: 999 Total (i.e. Null);  993 Residual
## Null Deviance:       10850000
## Residual Deviance: 823300    AIC: 9567
```

Notice that the solution selected by Optimus results in an RMSD of 28.693 which is lower than the RMSD of the Least Squares Solution (28.827) that assumes the appropriate model is $k_1x + k_2x^2 + k_3x^3 + k_4x^4$. Optimus selected a model which does not include all terms from the form used to generate the data. If a user were concerned by the fact that the model Optimus selected contains more terms (6) than are used in the representation of the de-noised data (4), the user could either increase the multiplicative factor associated with the term p in the AIC to more strongly penalise representations involving a greater number of parameters. Alternatively, the user could also modify the function `r()` to ensure that only a fixed number of terms are ever active.

Let us now take a look at how the adaptive thermoregulation performed given this highly non-smooth objective. The graphs below should now feel very familiar, they represent data taken from the last 20 000 iterations of the optimisation protocol executed by CPU 1.


System Pseudo Temperature (CPU 1)


Observed Acceptance Ratio (CPU 1)

Despite optimising a completely different model with a non-smooth objective function, the TCU succeeds in dynamically adjusting the system pseudo-temperature such that the observed acceptance ratio follows the annealing schedule rather well.

Acceptance Ratio Replica Exchange Optimus Run

Let us now consider the Acceptance Ratio Replica Exchange version of Optimus on 12 CPUs with the variable ACCRATIO defined as in **Tutorial 1**.

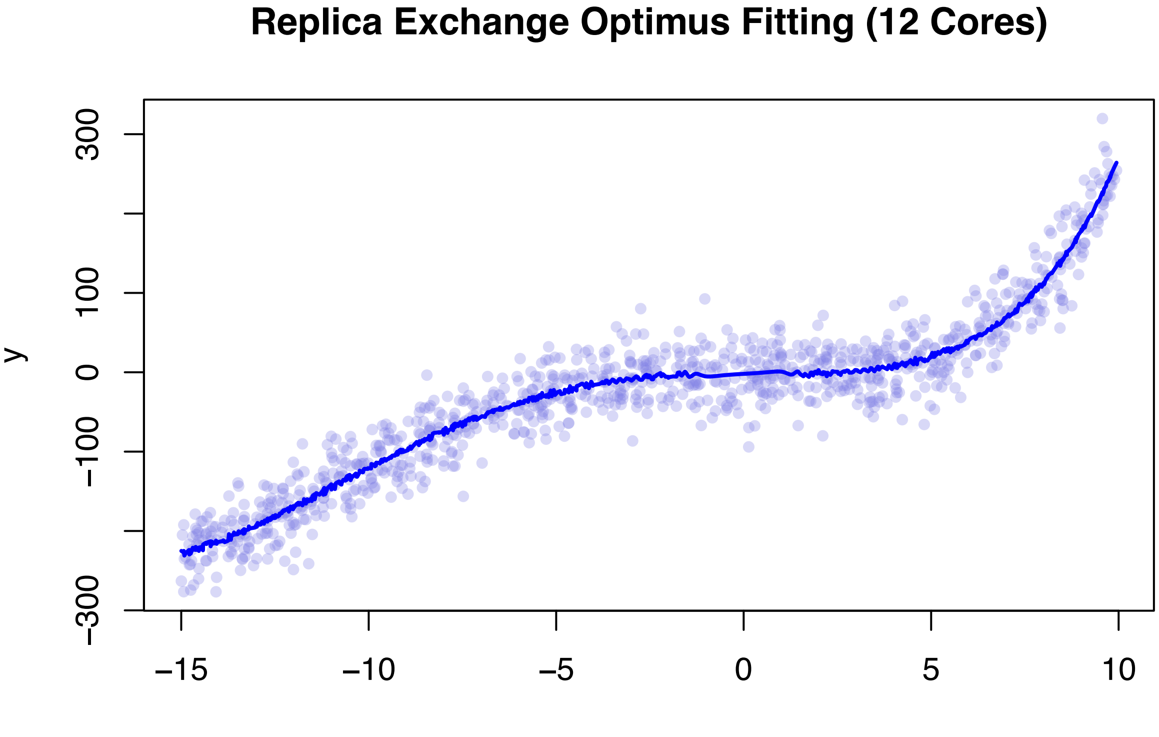
```
ACCRATIO <- c(90, 82, 74, 66, 58, 50, 42, 34, 26, 18, 10, 2)
```

As in the Acceptance Ratio Simulated Annealing run above, we will again execute the optimisation procedure for 200 000 iterations. The Replica Exchange version of Optimus takes an input argument EXCHANGE.FREQ (default value 1000) which specifies the total number of exchanges that will occur during the optimisation process. Consequently, the number of optimisation iterations that occur between subsequent exchanges between replicas can be calculated as NUMITER/EXCHANGE.FREQ, which is 200 iterations in this case.

Here we will set the input parameter `STATWINDOW` to have value 50 (its default value is 70). This signifies that the TCU will update the system pseudo temperature once every 50 iterations on each optimisation replica. This guarantees that 4 temperature adjustments will be made if a given replica is involved in two subsequent exchanges (because the number of iterations between exchanges is 200, as explained in the preceding paragraph) as opposed to merely 2 adjustments which would be the case if `STATWINDOW` were left to take its default value and could result in poor agreement between the observed acceptance ratio of the replica in question and the target acceptance ratio. The following line executes Optimus with the above specified inputs:

```
Optimus(NCPU=12, OPTNAME="term_12_RE",
        NUMITER=200000, STATWINDOW=50, DUMP.FREQ=100000, LONG=FALSE,
        OPT.TYPE="RE", ACCRATIO=ACCRATIO,
        K.INITIAL=K, rDEF=r, mDEF=m, uDEF=u, DATA=DATA)
```

Nine of the optimisation replicas (CPUs 1, 2, 3, 5, 6, 8, 9, 10 and 12) recovered the same solution that was found by the Acceptance Ratio Simulated Annealing Optimus run. Moreover, this solution is better (lower *AIC*) than those recovered by CPUs 4, 7 and 11. Thus, in this example, the Acceptance Ratio Simulated Annealing and Replica Exchange versions produce the same solution.



Please note that for convenience, only those replicas that produced a unique solution are listed in the table below (replicas 1, 2, 3, 6, 8, 9, 10 and 12 produced the same solution as replica 5; replica 11 produced the same solution as replica 7).

Table 5: 12-core Acceptance Ratio Replica Exchange results from Optimus run.

	CPU 4	CPU 5	CPU 7
Replica Acceptance Ratio	66.0000	58.0000	42.0000
E (AIC)	9.5673	9.5672	9.5673
Q (RMSD)	28.6664	28.6932	28.6951
Term 1	0.0000	0.0000	0.0000

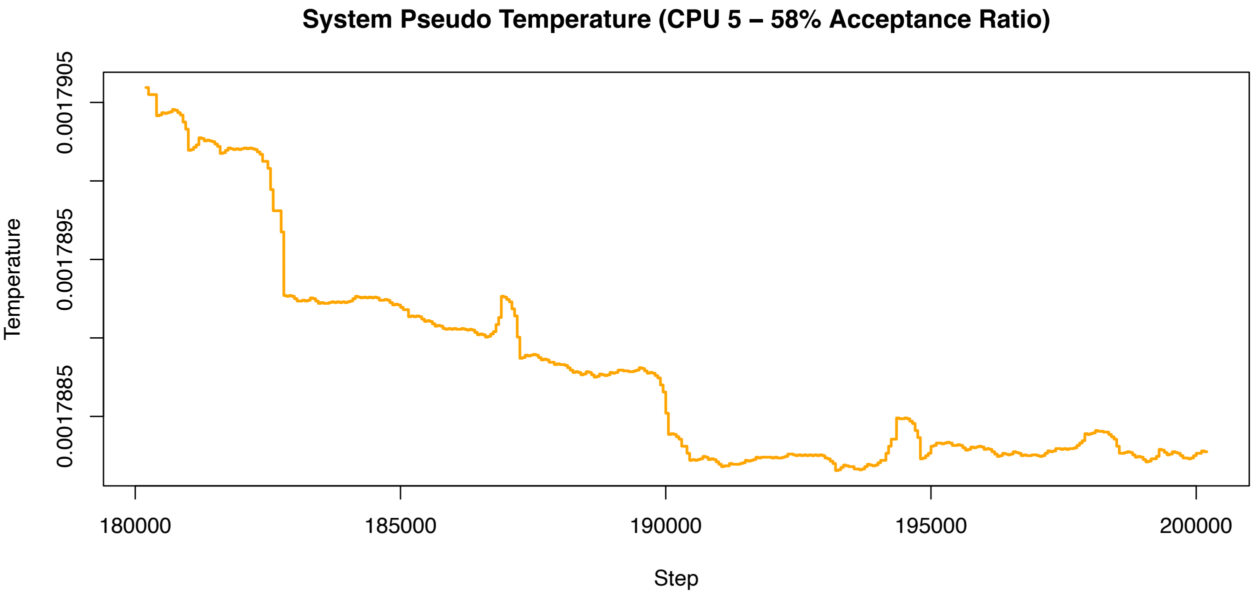
	CPU 4	CPU 5	CPU 7
Term 2	1.0000	1.0000	1.0000
Term 3	1.0000	1.0000	1.0000
Term 4	1.0000	1.0000	1.0000
Term 5	0.0000	0.0000	0.0000
Term 6	0.0000	0.0000	0.0000
Term 7	0.0000	0.0000	0.0000
Term 8	0.0000	0.0000	0.0000
Term 9	0.0000	0.0000	0.0000
Term 10	0.0000	0.0000	0.0000
Term 11	1.0000	1.0000	0.0000
Term 12	0.0000	0.0000	0.0000
Term 13	1.0000	0.0000	1.0000
Term 14	0.0000	0.0000	0.0000
Term 15	0.0000	0.0000	0.0000
Term 16	0.0000	0.0000	0.0000
Term 17	0.0000	0.0000	0.0000
Term 18	0.0000	0.0000	0.0000
Term 19	0.0000	0.0000	0.0000
Term 20	1.0000	1.0000	0.0000
Term 21	0.0000	0.0000	0.0000
Term 22	0.0000	0.0000	0.0000
Term 23	0.0000	0.0000	0.0000
Term 24	0.0000	0.0000	0.0000
Term 25	0.0000	0.0000	0.0000
Term 26	1.0000	1.0000	1.0000
Term 27	0.0000	0.0000	0.0000
Term 28	0.0000	0.0000	1.0000
Term 29	0.0000	0.0000	0.0000
Term 30	0.0000	0.0000	0.0000

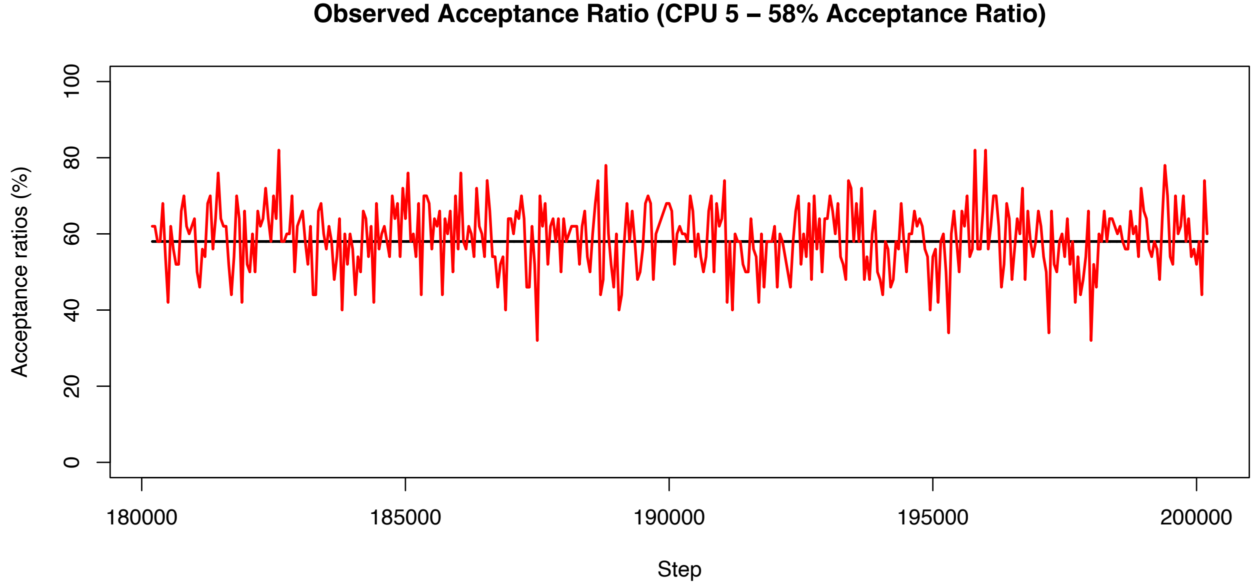
Note that the various replica outcomes illustrate the penalising effects of the AIC on models using a greater number of parameters. Consider the solutions found by the 66% acceptance ratio replica and the 58% acceptance ratio replica (CPUs 4 and 5 respectively). Let y_i denote the solution found by CPU i. Then, we have:

$$y_4 = b + c_2x^2 + c_3x^3 + c_4x^4 + c_{11}e^x + k_{13}c_{13}six(x) + c_{20}sin(x^3) + c_{26}cos(x^5)sin(-x)$$

$$y_5 = b + c_2x^2 + c_3x^3 + c_4x^4 + c_{11}e^x + c_{20}sin(x^3) + c_{26}cos(x^5)sin(-x)$$

Although the RMSD of y_4, 28.6664, is lower than the RMSD of y_5, 28.6932, y_5 has a lower value for AIC because it contains one less term than y_4. Since AIC was specified as the objective metric, Optimus (perhaps counterintuitively) selected y_5 as the more optimal solution to reduce overfitting.





The above graphs are produced using data from the last 20 000 iterations of the 58% acceptance ratio replica (CPU 5). It is clear that the observed acceptance ratio more strongly oscillated around the target acceptance ratio than was the case in the Acceptance Ratio Simulated Annealing run from the previous part of this tutorial. More generally, it should be expected that the observed acceptance ratio fluctuates more significantly around the target acceptance ratio in Replica Exchange than in Simulated Annealing, especially when the objective function is non-smooth as is the case in this example. This is because an exchange between two replicas has the same effect as restarting a Monte Carlo optimisation from a random initial configuration with a temperature that very likely is not conducive to the target acceptance ratio for the given configuration. As such, each time an exchange occurs, significant deviations from the target acceptance ratio may occur and may require several `STATWINDOW`s for the TCU to correct. Despite this challenge, the TCU performs satisfactorily.

Summary

We now understand how to employ Optimus to solve a more general problem than was addressed in **Tutorial 1** and one with a non-smooth objective function. Additionally, we have a better understanding of the adaptive thermoregulation. Using the Akaike Information Criterion (AIC) as a metric with which to evaluate the performance of a candidate model, taking into account the desire to represent the data while avoiding to overfit the data, both the Acceptance Ratio Simulated Annealing and Replica Exchange versions of Optimus recovered a better functional form to describe the data than the form which was assumed in **Tutorial 1** (based on how the data had been generated), obviously with some overfitting to adapt to the noise while with the stringency used in this example.

Table 6: Summary of solutions.

	E (AIC)	Q (RMSD)
Least Squares (Tutorial 1)	9.5705	28.82655
Optimus (AR Simulated Annealing)	9.5672	28.69324
Optimus (AR Replica Exchange)	9.5672	28.69324

Least Squares (**Tutorial 1**):

$$y = c_1 x + c_2 x^2 + c_3 x^3 + c_4 x^4$$

Optimus (Acceptance Ratio Simulated Annealing):

$$y = b + c_2 x^2 + c_3 x^3 + c_4 x^4 + c_{11} e^x + c_{20} sin(x^3) + c_{26} cos(x^5) sin(-x)$$

Optimus (Acceptance Ratio Replica Exchange):

$$y = b + c_2 x^2 + c_3 x^3 + c_4 x^4 + c_{11} e^x + c_{20} sin(x^3) + c_{26} cos(x^5) sin(-x)$$