List Solving vs. Procedure Function Case Study

A professor sent me a TK model with the following remarks…

“Todd, I have attached a code that is slow. On my machine it takes about 4 seconds for a solution, almost 40 for a list solve. The problem is that I have a call () statement that only needs to be executed once, it stays constant for the list solve. The method must work also when I use the optimizer. It seems to me that I have ran into this before but can’t remember the solution.”

Here is the variable sheet of his model.

Status	Input	Name	Output	Unit	Comment
	10	L			
	10	P			
	.1	ε_max			
	.4	height			
	.1	base			
	.8	n			
	1E7	K			
		Ω	7884.16949		
L	0	ξ			
L		δ	-.13096996		
L		δ_num	-.12959944		

Here is the rule sheet.

Rule
call Complementary()
Ω=2*base*K*(2/height)^n*((height/2)^(n+2)/(n+2))*(height/2
δ=-1/Ω^(1/n)*integral('integrand,0,L)
δ_num=-integral('integrand_num,0,L)

The model is set up to list solve with epsilon as the input list. The “Complementary” function fills some lists with 501 values each and only needs to be executed once for the given inputs on the variable sheet. It does not need to be repeated for each instance of list solving.

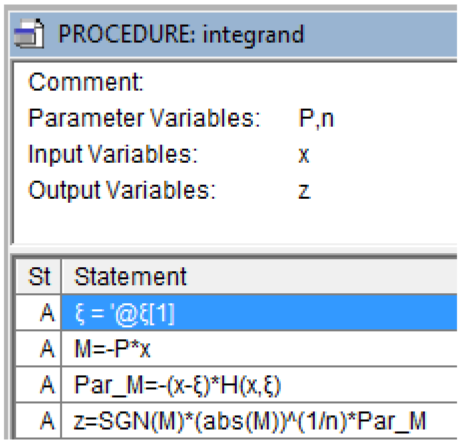
Statement
for i=1 to 501
'ε_max[i]=(i-1)*(ε_max/500)
'κ[i]='ε_max[i]*(2/height)
'M[i]=2*base*K*(2/height)^n*((height/2)^(n+2)/(n+2))*('ε_max[i])^n
next i
;
for i=1 to 501
'Complementary[i]=integral('dM_κ,0,'M[i])
next i
;
'dC_dM[1]=('Complementary[1+1]-'Complementary[1])/('M[1+1]-'M[1])
for i=2 to 500
'dC_dM[i]=('Complementary[i+1]-'Complementary[i-1])/('M[i+1]-'M[i-1])
next i

Function Complementary can be declared as a special Startup function in the Options menu. It will then be executed just once prior to any processing of the rule sheet, list solving, or optimizing. If the Optimizer is to be used, then List Solving must be avoided. The Optimizer cannot force multiple runs of the List Solver in the optimization process. It is better to replace the looping calculations done by List Solving with a procedure function. Here are the steps I took to convert the model.

Here is the procedure function used to loop through the 101 values of epsilon.

Statement
for i = 1 to length('ξ)
'@ξ[1] = 'ξ[i]
'δ[i] = -1/Ω^(1/n)*integral('integrand,0,L)
'δ_num[i] = -integral('integrand_num,0,L)
next i

The first statement in the loop takes the current value of epsilon and places it into the first element of a temporary list named @epsilon. The integrand and integrand_num functions need the value of epsilon but it cannot be passed to them as an input variable or a parameter variable so the trick is to use a list element instead. List elements are globally available within a model. Here is the integrand function.

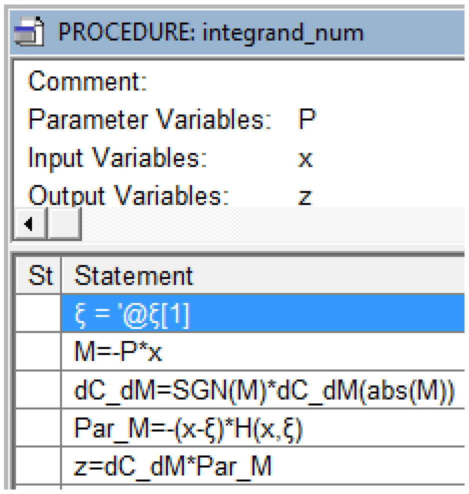


PROCEDURE: integrand

Comment:
Parameter Variables: P,n
Input Variables: x
Output Variables: z

St	Statement
A	ξ = '@ξ[1]
A	M=-P*x
A	Par_M=-(x-ξ)*H(x,ξ)
A	z=SGN(M)*(abs(M))^(1/n)*Par_M

The function must have only one input variable and one output variable so that it can be processed by the built-in integral function. Any other values must be passed into the function as either Parameter variables or list elements. Parameter variables P and n are used to get constants from the variable sheet. The value of epsilon is taken from the first element of the @epsilon list in the first statement. A similar process is used in the integrand_num function.


PROCEDURE: integrand_num

Comment:
Parameter Variables: P
Input Variables: x
Output Variables: z

St	Statement
	ξ = '@ξ[1]
	M=-P*x
	dC_dM=SGN(M)*dC_dM(abs(M))
	Par_M=-(x-ξ)*H(x,ξ)
	z=dC_dM*Par_M

The rule sheet is updated as follows.

Rule
Ω=2*base*K*(2/height)^n*((height/2)^(n+2)/(n+2))*(height/2)^n
; kept these for getting single value snapshots on the variable sheet
place('@ξ,1) = ξ ; to make the current value of ξ available to the integrands
δ=-1/Ω^(1/n)*integral('integrand,0,L)
δ_num=-integral('integrand_num,0,L)
call loop() ; replaces list solving

The rule sheet includes the omega calculation and then three rules to do the integrals using the single value of epsilon from the variable sheet. The place function is used to place that value of epsilon into the temporary list for access by the integrands.

The call to the loop function loops through the 101 epsilon values, replacing the List Solve process.

It is important to note that List Solving is still possible if the user of the model was interested in varying any of the inputs on the variable sheet – L, P, etc. The Optimizer is also still available for use as needed.

Finally, the model now runs through the complete list of solutions in 0.78 seconds on my PC – a significant improvement over the 40 seconds reported for the original version of the model.