

FlyingKoala

Bradley van Ree

Defining mathematical and technical models in MS Excel with Excel formulae but evaluating them using Python

	A	B	C	D	E	F	G	H	I	J	K	L
1	Time	Temp Min	Temp Max	Degree Day Equation 1	Degree Day Dynamic Array Equation 1	Degree Day Equation 2	Degree Day Dynamic Array Equation 2					
2	1/07/2018	-0.4	18.1	0	0	4.05	4.05					
3	2/07/2018	-2.6	21.2	0	0	5.6	5.6					
4	3/07/2018	-1.4	25.1	1.85	1.85	7.55	7.55					
5	4/07/2018	1.3	27.2	4.25	4.25	8.6	8.6					
6	5/07/2018	4.7	28.1	6.4	6.4	9.05	9.05					
7	6/07/2018	8.1	19.1	3.6	3.6	4.55	4.55					
8	7/07/2018	0.3	17.4	0	0	3.7	3.7					
9	8/07/2018	2	17.7		0		3.85					
10	9/07/2018	-1.1	17.8		0		3.9					
11	10/07/2018	-2	18.3		0		4.15					
12	11/07/2018	-1.9	18.3		0		4.15					
13	12/07/2018	-3.4	18		0		4					
14	13/07/2018	-3.8	20		0		5					
15	14/07/2018	-3.5	21		0		5.5					
16	15/07/2018	-2.2	22.6		0.2		6.3					
17	16/07/2018	1.3	24.2		2.75		7.1					
18	17/07/2018	1.7	25.8		3.75		7.9					
19	18/07/2018	1.7	26.2		3.95		8.1					
20	19/07/2018	3	26.5		4.75		8.25					
21	20/07/2018	4.7	18.9		1.8		4.45					
22	21/07/2018	-0.1	25.5		2.7		7.75					
23	22/07/2018	5.7	27.1		6.4		8.55					
24	23/07/2018	13	29.1		11.05		11.05					
25	24/07/2018	10.1	26.8		8.45		8.45					
26	25/07/2018	8.1	27.3		7.7		8.65					
27	26/07/2018	6.6	30.4		8.5		10.2					

For use with mathematical modelling where you want the advantages of Python calculation and/or libraries but your user base are non-technical

Contents

1	TL;DR	3
1.1	The problem	3
1.2	Features	3
1.3	Benefits	3
2	Why FlyingKoala?	4
3	About FlyingKoala	4
4	How do I make it work?	5
4.1	Setting up Excel	6
4.2	Setting up your model(/s)	6
4.3	Using your model(/s)	9
5	Under the hood (the code)	12
5.1	DegreeDay	12
5.2	DegreeDayDynamicArray	13
	Index	14

1 TL;DR

Srsly, I need my life back. Get to the point.

1.1 The problem

- Data analysis with interesting data sets (large or time series) is hard
- Excel can get in the way just as your data set becomes interesting
- Not everyone is going to learn to code, nor should they be expected to
- Domain experts and students are usually skilled enough in MS Excel
- Managers can't read code to see if a mathematical (/technical) model has been created correctly
- Auditing is difficult when everything is in code (They don't call it code for nothing!!)
- Scenario analysis usually requires a heap of overhead and is difficult to manage

1.2 Features

- MS Excel as a user interface to Python, notably Pandas (Dataframes), numpy, matplotlib, database connectivity, APIs like Harvest (timesheeting), Xero (accounting), etc...
- matplotlib graphs in Excel
- xlwings has a groovy SQL feature which does magical things
- Excel equations are defined in Excel but processed in Python with minimal code
- Results of Python calculations are available in Excel

1.3 Benefits

- The entire mathematical (/technical) model is available for managers to read because it's an Excel equation.
- Audits are easier because people know how to read and change Excel
- inter-company expressions of a calculation don't necessarily require all sides to have evenly skilled coders
- Makes time series data calculations with serious size data sets possible in Excel
- Makes big data calculations in Excel quicker
- Restricts the need for a coder geek from being involved in the model development process reducing time and errors
- Multiple mathematical models can be defined and assessed quickly. Great for scenario analysis.

2 Why FlyingKoala?

While doing data analysis work, particularly time series analysis (but not necessarily limited to), you very quickly find that the limits of Excel are much closer than you'd like.

The data analysis world has looked around for solutions and adapted the statistics powerhouse language R to Python forming the basis for the Pandas library. Due to the power of the Python community, matplotlib is often associated for the inevitable visualisations. And they are marvellous tools but are not necessarily intuitive.

There comes a challenge, however, when you have a number of experts who aren't necessarily code savvy but are still required to do some number crunching. These people could be engineers, finance (accountants, book keepers), sales people, specialists in quite unrelated fields like biology, botany, research or even students in essentially any field. Often enough these people are adept at Excel but are, understandably, not going to become proficient in writing computer code.

It would be great if there were a solution which could take advantage of the abundantly available Excel skills and minimise the amount of computer code. This would put the domain specialists in more direct contact with the models they are developing, without insisting on having them learn to code or, often fraught with errors and delays, having a code savvy person translate the domain specialists model.

FlyingKoala addresses this problem. It enables non-coding domain specialists to leverage the large and well supported data analysis toolkit(/s) found within Python.

3 About FlyingKoala

FlyingKoala is an integration of two Python projects; Koala2 and xlwings.

If things start sounding too technical, skip to the next paragraph - you won't really miss too much.

The advantage of using both of these projects in conjunction with each other is that you can move performant mathematical model creation from the hands of people who know how to code to domain specialists who essentially only know MS Excel.

From the Koala2 documentation:

Koala converts any Excel workbook into a python object that enables on the fly calculation without the need of Excel.

Koala parses an Excel workbook and creates a network of all the cells with their dependencies. It is then possible to change any value of a node (an "Excel cell") and recompute all the depending cells.

The outcome of this is that Koala2 can read a workbook and understand how the workbook cells relate to each other. If we use named ranges and named cells, we can teach the workbook the language of your mathematical or technical model. The smart use of named ranges enables Koala2 to figure out which cells are related to a particular mathematical or technical model in a way we can find the cells later on. FlyingKoala knows how to find the named cells and then put new values in the various input cells. FlyingKoala then tells the Koala2 spreadsheet to evaluate an equation.

From the xlwings documentation:

xlwings is a BSD-licensed Python library that makes it easy to call Python from Excel and vice versa:

Scripting Automate/interact with Excel from Python using a syntax close to VBA.

Macros Replace VBA macros with clean and powerful Python code.

UDFs Write User Defined Functions (UDFs) in Python (Windows only).

REST API Expose your Excel workbooks via REST API. Numpy arrays and Pandas Series/DataFrames are fully supported.

xlwings-powered workbooks are easy to distribute and work on Windows and Mac.

The outcome of this is xlwings takes care of the communication between Excel and Python. User Defined Functions (UDFs) are the main function FlyingKoala uses, but the other functions that come with xlwings are absolutely fantastic for data analysis, data migration, data visualisation, accounting integrations and other work.

FlyingKoala uses xlwings' ability to manage the communication between Python and Excel through which the access to a full Python environment becomes really easy. FlyingKoala then takes advantage of Koala2's conversion of equations, and all associated inputs (cells), into Python which can then be applied to cells in Excel.

4 How do I make it work?

The worked example uses degree days as the mathematical/technical subject. Degrees day calculation is easy and there are a multitude of ways to calculate them. The data we are using is a weather time series from the Australian Bureau of Meteorology for Alice Springs Airport 2018-07-01 through 2019-06-31.

The two growing degree day calculations we will use come from Wikipedia and are defined;

$$GDD = \frac{T_{\max} + T_{\min}}{2} - T_{\text{base}} \quad (1)$$

$$GDD = \max\left(\frac{T_{\max} + T_{\min}}{2} - T_{\text{base}}, 0\right) \quad (2)$$

Using FlyingKoala there is very little to see from a technical perspective. The “secret” to FlyingKoala, from a user perspective, is clever use of what Excel calls named ranges or named cells.

4.1 Setting up Excel

We will step through constructing an Excel spreadsheet. When it comes to constructing a spreadsheet for your own work you are welcome to stray from the following structure.

Ensure your computer has Python installed (Anaconda is supported) and MS Excel. In the Python environment ensure you have installed xlwings, Koala2, Pandas and numpy.

Create a workbook and add the xlwings plug-in. The best way I found is to create one by using the xlwings quick way but at the end of the day xlwings is simply a plug-in like any other Excel plugin.

Due to xlwings, you are going to start with a tab `_xlwings.conf` and Sheet1.

In a new workbook make the following tabs:

Raw Data Your un-processed data

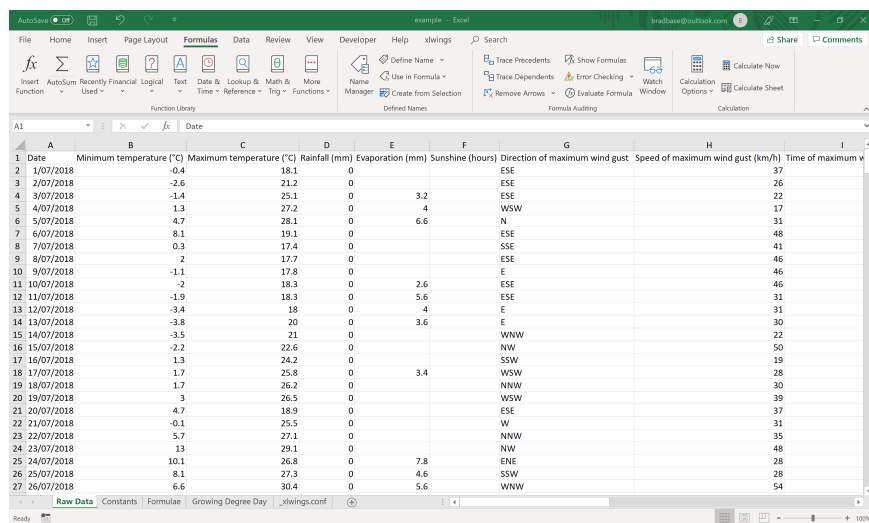
Constants A worksheet to contain values used in calculation which will remain “constant”.

Formulae The worksheet where your mathematical models will be defined

Growing Degree Day The worksheet where the results of calculation will be

4.2 Setting up your model(/s)

The Raw Data tab is currently ignored by FlyingKoala. This will be discussed when we look at the code.



Date	Minimum temperature (°C)	Maximum temperature (°C)	Rainfall (mm)	Evaporation (mm)	Sunshine (hours)	Direction of maximum wind gust	Speed of maximum wind gust (km/h)	Time of maximum wind gust
1/07/2018	-0.4	18.1	0			ESE		37
2/07/2018	-2.6	21.2	0			ESE		26
3/07/2018	-1.4	25.1	0		3.2	ESE		22
4/07/2018	1.3	27.2	0		4	WSW		17
5/07/2018	4.7	28.1	0	6.6		N		31
6/07/2018	8.1	19.1	0			ESE		48
7/07/2018	0.3	17.4	0			SSE		41
8/07/2018	2	17.7	0			ESE		46
9/07/2018	-1.1	17.8	0			E		46
10/07/2018	-2	18.3	0	2.6		ESE		46
11/07/2018	-1.9	18.3	0	5.6		ESE		31
12/07/2018	-3.4	18	0		4	E		31
13/07/2018	-3.8	20	0	3.6		E		30
14/07/2018	-3.5	21	0			WNW		22
15/07/2018	-2.2	22.6	0			NW		50
16/07/2018	1.3	24.2	0			SSW		19
17/07/2018	1.7	25.8	0		3.4	WSW		28
18/07/2018	1.7	26.2	0			NNW		30
19/07/2018	3	26.5	0			WSW		39
20/07/2018	4.7	18.9	0			ESE		37
21/07/2018	-0.1	25.5	0			W		31
22/07/2018	5.7	27.1	0			NNW		35
23/07/2018	13	29.1	0			NW		48
24/07/2018	10.1	26.8	0	7.8		ENE		28
25/07/2018	8.1	27.3	0	4.6		SSW		28
26/07/2018	6.6	30.4	0	5.6		WNW		54

Figure 1: Raw Data

After reading our Growing Degree Day article on Wikipedia we can see there is a term called T_{base} (the base or reference temperature) and, according to the article, this value is usually defined as 10°C but this is open to change. So we will consider this a constant.

	A	B	C	D	E
1	Constants				
2					
3	T_{base}	10			
4					
5					

Figure 2: T_{base}

As can be seen in cell Constants!B3, we have the value 10. The cell Constants!B3 is a named range and it is called T_{base} .

Now for the Formulae tab. This is where the two formulas will be defined. The two formulae have a few variable terms in common, namely T_{min} and T_{max} . No prizes for figuring these are the daily minimum and maximum temperatures in degrees Celsius.

The cell Formulae!C5 will be the the maximum temperature and Formulae!D5 will be the minimum. Again, both are named ranges.

- Formulae!C5 is called T_{max}
- Formulae!D5 is called T_{min}

	A	B	C	D	E
1	Formulae				
2					
3	Growing Degrees Day				
4			T_{max}	T_{min}	
5	Equation 1	4.25	1.3	27.2	
6	Equation 2	4.25			
7					
8					
9					

Figure 3: T_{max}

Label the variables (eg; something useful in Formulae!C4 and Formulae!D4).

Function Library					
T_min				27.2	
	A	B	C	D	E
1	Formulae				
2					
3	Growing Degrees Day				
4			Tmax	Tmin	
5	Equation 1	4.25	1.3	27.2	
6	Equation 2	4.25			
7					

Figure 4: T_{\min}

We will now define each of the Growing Degree Day models we want to explore. Be sure to use the named range of the variables and constants.

The use of named ranges and cell names is actually the “secret” behind making FlyingKoala work. By creating and using named ranges and cell names you are actually teaching Excel the language of your mathematical or technical model and Koala2 uses this acquired language to figure out how to run your calculations.

Equation 1, from earlier, will be placed into Formulae!B5. In Excel’s formula it looks like this:

$$=MAX(((T_{\max}+T_{\min})/2)-T_{\text{base}}, 0)$$

Label the equation (eg; something useful in Formulae!A5)

Function Library						
Equation_1				=MAX(((T_max+T_min)/2)-T_base, 0)		
	A	B	C	D	E	F
1	Formulae					
2						
3	Growing Degrees Day					
4			Tmax	Tmin		
5	Equation 1	4.25	1.3	27.2		
6	Equation 2	4.25				
7						

Figure 5: Equation (1)

Equation 2, from earlier, will be placed into Formulae!B6. In Excel’s formula it looks like this:

$$=IF(T_{\min} < T_{\text{base}}, ((T_{\max}+T_{\text{base}})/2)-T_{\text{base}}, ((T_{\max}+T_{\min})/2)-T_{\text{base}})$$

Label the equation (eg; something useful in Formulae!A6)

Function Library					Defined Names				
Equation_2					=IF(T_min < T_base, ((T_max+T_base)/2)-T_base, ((T_max+T_min)/2)-T_base)				
	A	B	C	D	E	F	G	H	I
1	Formulae								
2									
3	Growing Degrees Day								
4			Tmax	Tmin					
5	Equation 1	4.25	1.3	27.2					
6	Equation 2	4.25							
7									

Figure 6: Equation (2)

In both cases of Excel expressions for the equations we can see that;

- Formulae!B5 is a named range called Equation_1
- Formulae!B6 is a named range called Equation_2

4.3 Using your model(/s)

We will be using the worksheet Growing Degree Day. For this there is a need for some data and then we can use a User Defined Function to operate on the data.

- “Growing Degree Day”!A will be a label, in this case dates.
- “Growing Degree Day”!B will be the minimum temperature.
- “Growing Degree Day”!C will be the maximum temperature.

“Growing Degree Day”!D will a User Defined Function (UDF). This does require *some* code (but not much and is usually boilerplate / copy+paste). The code will be discussed later. The DegreeDay UDF is written in Excel as;

=DegreeDay(Equation_1,\$B2,\$C2)

Walking through the UDF called DegreeDay it presents as an integrated Excel formula. It even comes up as an option when you start typing. The code for DegreeDay function, however, is written in Python and executes in Python.

The first argument for the DegreeDay function is the named range for the model we want to evaluate. In the above example this is Equation_1.

The second argument for the DegreeDay function is the minimum temperature. We can put a number in there but we are in Excel and would like to use the drag-down action so we will use a cell reference.

The third argument for the DegreeDay function is the maximum temperature. As with the minimum temperature we can put a number in there but we are in Excel and would like to use the drag-down action so we will use a cell reference.

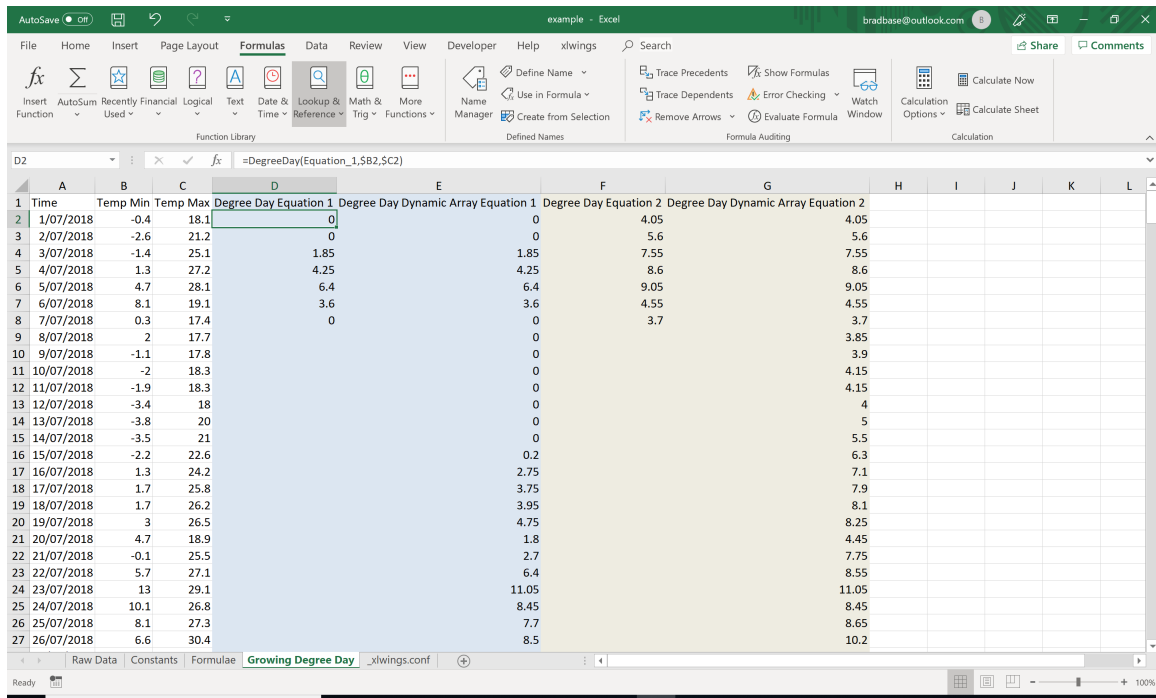


Figure 7: Degree Day UDF Equation (1)

Like all functions in Excel, when you type it up right and hit enter, it'll evaluate and put the result in the cell you have defined it in. And you are welcome to do a fill-down to see it calculate for the entire series.

Now for the leap...

If you change the arguments on the DegreeDay UDF, you'll be using the second equation we had defined.

`=DegreeDay(Equation_2,$B2,$C2)`

Try it. Think about it. Enjoy it for a second.

This particular user defined function is great. We can see that FlyingKoala is using the same inputs on different equations and providing results. But, I'm certain you have already found, a fill-down for a years worth of data very quickly slows your computer down.

Considering we may well be doing a lot of time series analysis we would not be able to live long enough for some calculations. Thankfully Excel provides a solution in the form of Dynamic Arrays. It's actually a solution that's been there a long time and can be used outside the context of FlyingKoala but very few people know how to take advantage of it. Thankfully, we can continue living in ignorance as FlyingKoala (actually, xlwings) has our back. We have available to us a second UDF which takes advantage of Dynamic Arrays. Creatively it's called DegreeDayDynamicArray.

`=DegreeDayDynamicArray(Equation_1,$B2:$B366,$C2:$C366)`

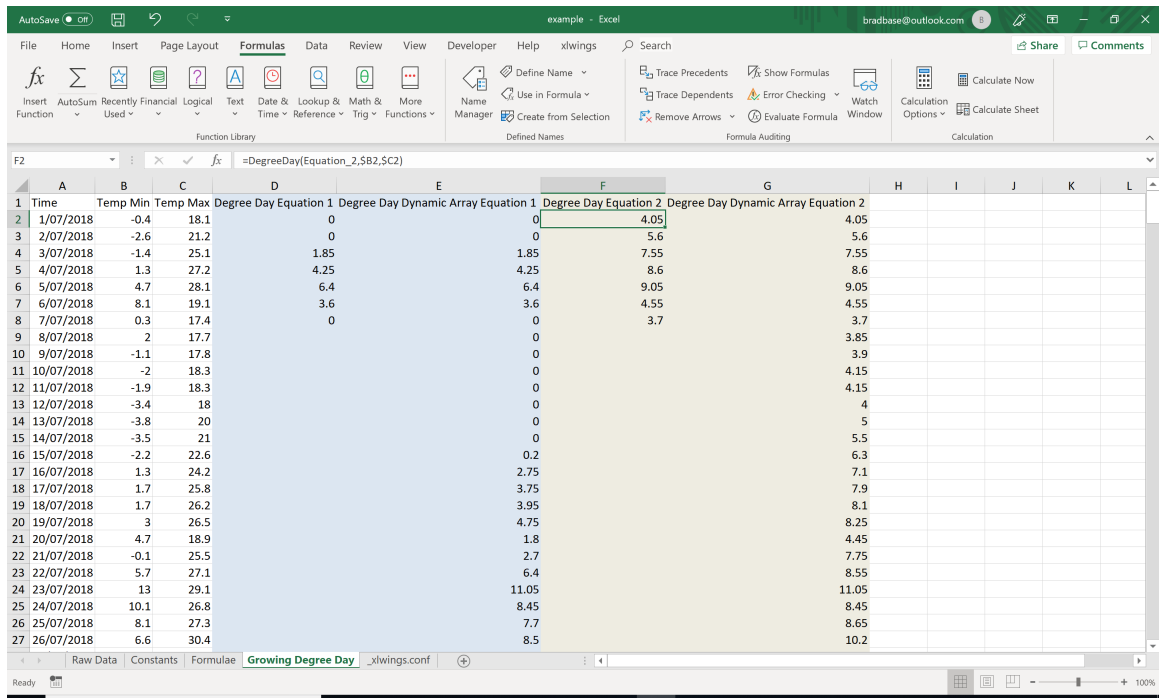


Figure 8: Degree Day UDF Equation (2)

The only real usability difference between this function and the previous one is we can supply a range for the input of both the minimum and maximum temperatures. Providing the inputs are the same “shape” (they have the same number of rows), the result is an automatically filled in Dynamic Array with the results of the Equation_1 for the series.

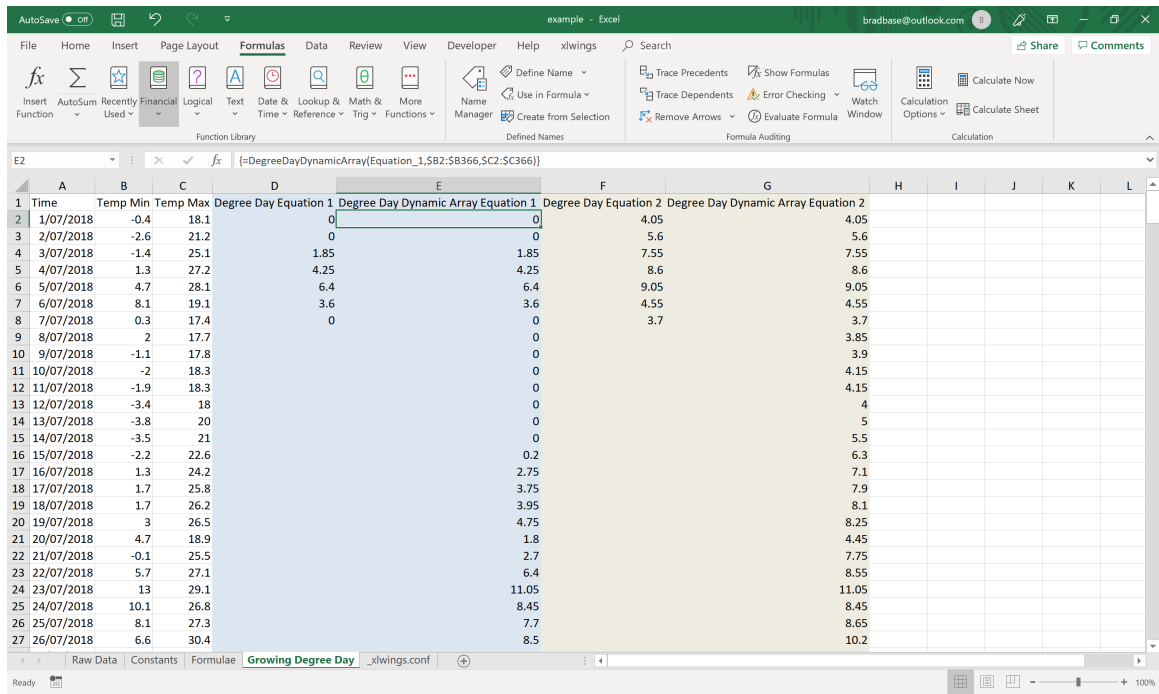


Figure 9: Dynamic Array Equation (1)

We can do the same for Equation_2

=DegreeDayDynamicArray(Equation_2,\$B2:\$B366,\$C2:\$C366)

The Dynamic Array technique calculates results *really* quickly and can easily handle very big time series.

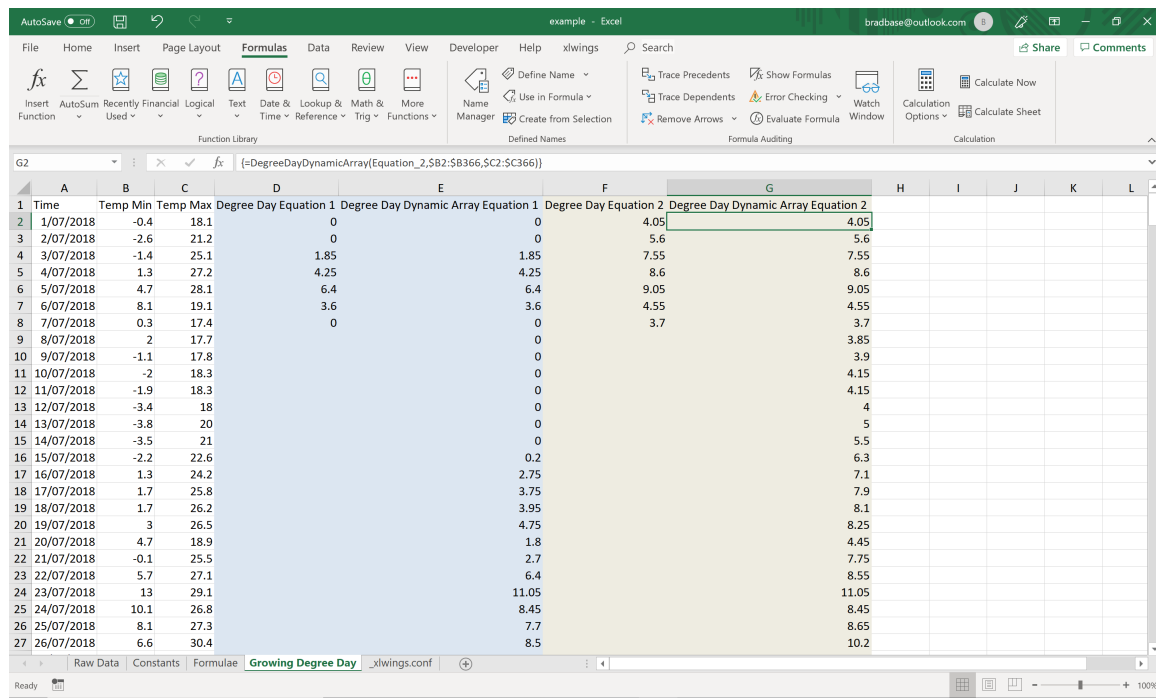


Figure 10: Dynamic Array Equation (2)

5 Under the hood (the code)

I have mentioned the code a couple of times. It is usually required to write a few UDFs as your domain is different from other people's. I am working on making FlyingKoala come with a collection of common ones but that's down the road a little and will require users to submit the ones they've written.

From here I'll assume you know Python or are near someone who can fill you in on the gaps.

5.1 DegreeDay

The DegreeDay user defined function (UDF) is very literally a UDF as described by xlwings. If you are interested in writing UDFs I'll leave you to read from the xlwings doco. But we can see roughly 5 lines of code with a further 5 lines of mark-up.

```

1 @xw.func
2 @xw.arg('model', xw.Range, doc='Name, as a string, of the model which will
   be evaluated. The Excel cell name / named range')
3 @xw.arg('T_min', np.array, doc='Daily minimum temperature')
4 @xw.arg('T_max', np.array, doc='Daily maximum temperature')
5 @xw.ret(index=False, header=False)
6 def DegreeDay(model, T_min, T_max):
7     """Function to assemble a dataframe for calculating Degree Day"""
8
9     if not isKoalaModelCached(model.name.name):

```

```

10     generateModelGraph(model)
11
12     inputs_for_DegreeDay = pd.DataFrame({'T_min': np.array([T_min]), 'T_max'
13     : np.array([T_max])})
13     return EvaluateKoalaModel(model.name.name, inputs_for_DegreeDay)

```

5.2 DegreeDayDynamicArray

Same goes for the DegreeDayDynamicArray UDF. As the terms T_min and T_max come in as numpy arrays we don't need to cast them.

```

1 @xw.func
2 @xw.arg('model', xw.Range, doc='Name, as a string, of the model which will
   be evaluated. The Excel cell name / named range.')
3 @xw.arg('T_min', np.array, doc='Daily minimum temperature')
4 @xw.arg('T_max', np.array, doc='Daily maximum temperature')
5 @xw.ret(index=False, header=False, expand='down')
6 def DegreeDayDynamicArray(model, T_min, T_max):
7     """Function to assemble a dataframe for calculating Degree Day using
   dynamic arrays"""
8
9     if not isKoalaModelCached(model.name.name):
10         generateModelGraph(model)
11
12     inputs_for_DegreeDay = pd.DataFrame({'T_min': T_min, 'T_max': T_max})
13     return EvaluateKoalaModel(model.name.name, inputs_for_DegreeDay)

```

The key to a UDF is;

- Unique name. You'll get bizarre errors if the UDF name is the same as a named range.
- model is always the first argument
- One parameter per term in your equation

Obviously they can get more sophisticated. Calling out to databases, APIs, casting types, doing all sorts of other manipulations. It is Python so you've got everything here you'd want.

Index

T_{base} , 7

Constant, 7

DegreeDay, 9, 10

DegreeDayDynamicArray, 10, 12

Equation (1), 5, 8

Equation (2), 5, 8

Equation_1, 8

Equation_2, 8

Formulae, 7

Koala2, 4

model, 8

Raw Data, 6

T_base, 7

T_max, 7

T_min, 7

xlwings, 5