Gerald's Column
by Gerald Fitton

Thanks again for all the many interesting letters and emails. You can write to me at the address given in Paul's Fact File for Abacus Training. Please note that my Post Code is changing from SN2 6QA to SN2 7QA; of course either post code will reach me during the change over period. If you have a problem then please do send me an example file rather than a lengthy description. Self addressed sticky labels do help more than you might guess and return postage is more than welcome!

This month I have received more emails about financial functions than usual. Perhaps it is because we have just passed the end of the Financial Year! However, I am not going to deal with Tax Return Applications (by the way, Rex Palmer has a good one which runs in both Fireworkz for RISC OS and Fireworkz for Windows). Instead I shall explain something about Mortgages and Annuities and use my example to demonstrate a simple iterative technique called the 'Binary Chop'.

Help Wanted

A full set of PipeDream and Fireworkz files, including an iterative custom function, appear on the Archive monthly disc and elsewhere. I had intended to include something similar for Schema but, owing to a pressing invitation to my grandson's ninth birthday I have had to limit my activites so, with regret, no example for Schema users (this month).

In the past whenever I have remarked that I needed help from you with creating a Schema macro I've always had more than one offer. So, here's the deal. If you would like to see your name in print then let me have a similar set of Schema files including the essential iterative macro. Please include an explanation of how your set of files works and comment on problems which you have had creating the set of files. I will include it as soon as I can.

Introduction

It was quite a while ago now that Bryn Fursman first asked me for help with converting some financial programs he had written in BASIC so that they would run as a PipeDream spreadsheet. The full set of BASIC programs are included on the Archive monthly disc; you'll find them in the BASIC sub directory together with a [!ReadMe] file describing how to use them. I believe that you'll find them interesting in their own right.

Mortgages and Annuities

Suppose you borrow some money and agree to pay back a fixed amount each month for many years—for example, repaying a mortgage. The person lending you the money expects you to pay interest on the outstanding part of the money you've been lent. It is this interest which makes the amount you pay back greater than the amount borrowed.

Let's take another scenario. In this new scenario called an Annuity you give money to a financial organisation. In return they agree to give you a certain amount of money every month for many years—but they don't return the capital to you at the end of the term.

I am sure that you will see that this case is exactly the same as the first case but the roles have been reversed. Instead of being the borrower you have become the lender.

When doing sums on repayment mortgages or fixed term annuities the usual thing to calculate is the size of the monthly payments. Everything else (the initial amount, the rate of interest, the number of years) has known values. The formula for working out the size of the monthly payments is an explicit formula—explicit means that you substitute known values for the variables in the right hand side of an equation and the answer appears. Spreadsheet cells always contain explicit formulae.

The Iterative Method

There is no explicit formula for calculating the interest rate of a mortgage or annuity when you know all the other variables. The only way of finding the answer is to use an iterative method in which you make a guess and then see if it was the right one.

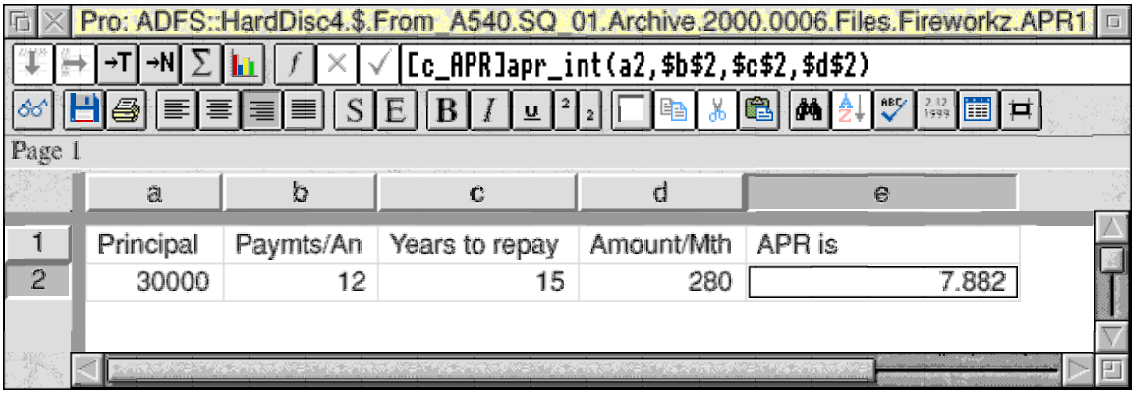
Bryn said that it was calculating the APR (which stands for Annual Percentage interest Rate) which was giving him the most trouble—he had something which he said ‘sort of’ worked but took far too long to get the answer to two decimal places. The reason for his difficulty is twofold. Firstly you can not find the value of the APR with what mathematicians call an ‘explicit formula’. The solution requires iteration. Secondly Bryn’s method of iteration is one which does not converge rapidly.

I developed a custom function for Bryn which runs in an acceptable time using a technique known as the ‘binary chop’! This particular technique may be one which you will find useful for non financial applications such as engineering or even astronomy!

Using this Fireworkz Application

Let’s s see how the Fireworkz version works.

Load the file [APR1] and you’ll find that the custom function [c_APR] will also load.



The values in A2 (the Principal), B2 (the number of payments each year), C2 (the number of years) and D2 (the monthly repayments) are all given values. What we have to find is the interest rate (APR). It is found by the iterative custom function shown in E2.

You can change any of the values in the block A2D2 and the custom function will return the corresponding interest rate. In the example I have chosen a £30 000 mortgage with repayments of £280 a month for 12 years. The interest rate returned is 7.882%

Present Value

Let me introduce you to the term 'Present Value of an Annuity'. Remember that the Mortgage scenario where you borrow money is simply the mirror image of an Annuity.

If you are given the interest rate, the monthly payments you want to receive and the number of years the annuity will last, then, using an explicit function, you can work out how much you would have to invest in order to ensure that the annuity is paid. The amount is called the 'Present Value of the Annuity'. In the example shown in the screenshot the Present Value of the proposed Annuity is £30 000.

Finding the APR

In Bryn's APR problem he knows the Present Value, as well as the value and the number of monthly repayments. It is the interest rate, the APR, which is unknown.

A 'guess and see' iterative method must be used because there is no explicit formula. First you guess an interest rate and work out a Present Value (PV). If the calculated PV is higher than the initial amount you actually paid for your annuity then you know that you've chosen an interest rate which is too low. Guess again with a higher figure. Continue until the PV you calculate is 'near enough' the amount you've actually paid.

I have used the custom function [c_APR] to execute this iteration. With some reluctance I have decided not to show the custom function in print in the magazine. If you would like a copy and can not obtain the disc files elsewhere then please let me know and I'll help you.

Leaving the details out of the printed magazine is somewhat of an experiment and I would like to know whether you approve or not.

Referring now to the custom function file [c_APR].

Lines 23 to 35 of [c_APR] is a Repeat – Until loop which iterates as follows:

First we have to guess a couple of interest rates. I have chosen 0% and 900% as my lower and upper limits. If the custom function returns an error message (or worse) then you will need to change one or both of these guesses. You'll find them in rows 18 & 19.

These two guesses are averaged to give an initial value of 450%. The initial average rate is calculated in row 20 (outside the iterative Repeat – Until loop) but when going around the loop the average value is found in row 32.

The average interest rate is used to calculate the PV. The PV is subtracted from the initial amount to obtain the variable which I've called "pr_va". The formula which does this in row 25. What we are trying to find is the interest rate for which "pr_va" is zero. We can never find it exactly but we can find an approximation which is as close as we wish.

Now back to the two original guesses for the interest rate. One of them is replaced by the average interest rate. It is the higher of the two which is replaced if the PV is higher than the Principal (when "pr_va" is positive); the lower value is replaced if the PV is lower than the Principal (row 29). At every stage of the iteration one of the two 'guesses' will be retained and the other replaced by the previous average rate. The value we want to find always lies between the two current 'guesses'—this allows us to estimate the error.

At each stage of the iteration (the Repeat – Until loop of rows 23 to 35) the guesses get closer together; the difference between the pair of guesses is halved each time.

As with all iterative processes we have to define the exit condition. The exit condition I have defined in row 35 is that the two guesses for the interest rate have a difference between them of less than 0.001%. The accuracy of the answer is about three decimal places. If you wish, you can improve this accuracy by changing the value in row 35 to something smaller. Each extra decimal place is worth about three more times around the loop. As written, the custom function executes the loop about 17 times.

Varying the Years

The file [APR2] is a variant on [APR1] which calculates the effect of phasing an identical monthly payment of £280 over a period of between10 to 29 years. The APR varies from just over 2.3% up to just over 11.2%.

Varying the Payments

Similarly in the file [APR3] I have varied the payments from £150 per month to £340 per month using a fixed period of 20 years to pay back the loan.

Other Variants

I'm sure that you can think of other variants which might be useful such as varying the initial sum borrowed or paying quarterly instead of monthly.

Summary

One simple type of iteration technique is to keep halving the difference between two guesses. There are other much more sophisticated methods of iteration which will converge more rapidly (and hence speed up the calculation) but this method is one which people find relatively easy to understand. An advantage of using an understandable method is that you'll recognise when it won't work!

Let me give you just one example of the type of problem where this halving won't work so that you'll see what I mean. In the problem I have described you have to find when a certain expression is zero. Whenever you raise the interest rate the PV also increased. The PV is a 'monotonic' function of the interest rate. If the expression is not 'monotonic' then sometimes increasing the 'independent variable' increases the answer and sometimes it reduces the answer. You can't use the answer to decide which guess to replace.

Nevertheless, the method I've describe is simple and useful. I recommend it to you as the simplest choice whenever you have to solve by iteration. The two initial guesses must be chosen so that the answer must lie between them. The only limitation of the 'binary chop' technique is that the function must be monotonic between the two initial guesses.