

Loan Repayment Schedules using Equal Total Payments Method: Application for Financial Services

Mobin M. Idrees

University of Leicester
Department of Computer Science
Business Information Systems CO7518
Assignment 2

1. INTRODUCTION

In financial service practice the amortization schedule is a useful tool to determine the payback schedule of loans on an agreed term. The Equal total payment is an ideal method for the banks so that the banks could recover payment on high interest rate to make early profits. The objective of this work is to learn and implement algorithms for equal total payments in order to create an application for generating an amortization schedule.

The application is developed using Hyper Text Markup Language (HTML), Cascading Style Sheets (CSS), JavaScript and JQuery. The reason is to develop an application independent of frameworks or runtime environments. Utilized my learning from re-engineering module, I tried to eliminate the bad smells from the code like code duplication, used modularization, used comments and tried the source code to be self explanatory. [\[Source Code\]](#)

For the User Interface Design, the design is simple but interactive, I have used pie and bar graphs. Feedback principal has been utilised to reflect immediate repose from the user. Using the visibility principal the desing is not distractive and the users can navigage easily to what information they are looking. [\[User Guide\]](#)

2. AMORTIZATION METHODS

There are several methods used to make an amortization schedule depends on the requirements and nature of the loan. Most common forms are Mortgage or Car loan. This could be classified in either short or long term. These most common methods include:

- Total Equal Payment
- Equal Principal Payments
- Revolving
- Bullet Method
- Balloon Mortgages.

3. ASSUMPTIONS

- The decimal values varies in different stages of calculation, for instance, the **parseFloat()** supports **18** digit decimals while the **Math.pow()** returns **13** decimals digits. Therefore, the outstanding balance may appear in negative, however the variance of the results is not more than **+/-0.01%**

```
PP = (Rn * A) * Math.pow((1 + Rn), N) / (Math.pow((1 + Rn), N)-1);  
Rn = parseFloat($('input[name=interest]').val())/100/12;  
PP 3019.5901804309974  
Rn 0.004583333333333333
```

- The currency used is **'£'** which means that the application is assumed to be used in the United Kingdom and the nearby regions.
- It is assumed that the application is to be used as a standalone calculator and a schedule generator, as there is no option to save the results or integrate it with other applications.
- Fixed interest rate has been used for the term and the time value of the money is not considered, therefore the payment remains the same and the principal amount and the interest varies throughout the term.
- During the calculations stage no rounding of decimal values has been used, however the rounding option is used only for the final results.
- The Loan Period must be an integer value and cannot be alphabet or decimal. The accepted values are between **1** to **1200** months or **100** Years, while the maximum loan amount could be **1** billion.

4. IMPLEMENTATION

For simplicity and possibility to run the application independent of external frameworks and components, I decided to implement browser base application. Included JQuery for chart and amortization table and included CSS in a separate script file, the following code used in HTML to define external javascript files.

```
<script type="text/javascript" src="js/jquery.min.js"> </script>  
<script type="text/javascript" src="js/amortize.js"> </script>  
<script src="js/highcharts.js"> </script>
```

The following figure demonstrate the flow chart of the execution process for computing monthly amortization schedule. After taking the amount period and interest rate the **R_n** is calculated which is the monthly rate of interest. The loop starts from **1** to **N** and inside of the loop principal amount, interest amount paid and outstanding balance is calculated and update in the amortization table. After the execution of the loop the charts are updated on the screen.

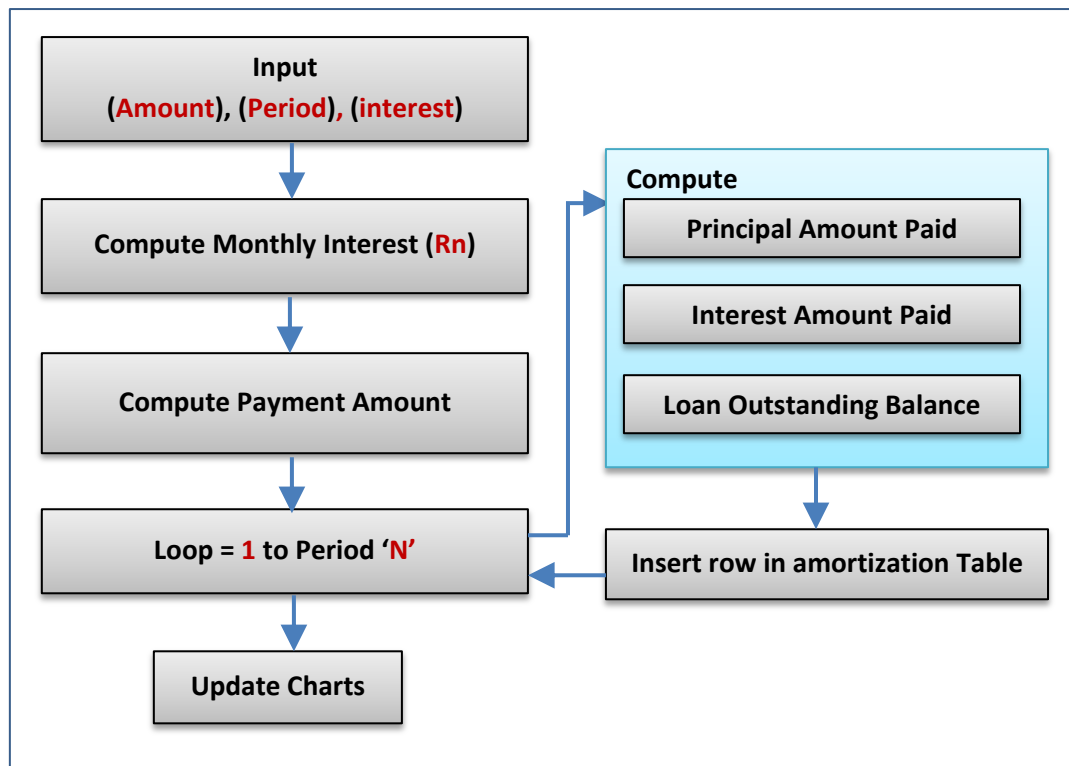


Figure 1: Overall Flow of Process

Upon submission the form by pressing the Calculate button, the data is moved to the JQuery functions available in the "amortize.js". Several tasks are performed that, includes "loadData", "computeMonthly" and updateGraph" as shown in the code below. [Download file](#)

```

// JQuery functions in Amortize.js
$(function () {
  $('#Calculate').click(function(){
    loadData();
    computeMonthly(A, N, Rn, PP);
    clearTable("alternatecolor");
    updategraph(A,PP,N);
  });
});

```

```
// Load data into variables
function loadData(){
    A = parseFloat($('input[name=amount]').val());
    N = parseFloat($('input[name=period_m]').val());
    Rn =parseFloat(($('input[name=interest]').val())/100)/12; ...}
```

Furthermore, the Monthly Payment “**PP**” is calculated using “Math.pow” library function and “updateFields” method is called to update the user interface as shown in the following code

```
// Calculate Monthly Payment using fixed payments
PP = (Rn * A) * Math.pow((1 + Rn), N) / (Math.pow((1 + Rn), N)-1);
updateFields(A, N, PP);
```

The updateFields” method along with calling the “format_currency” function updates the user interface to reflect the summarized loan information as shown in Figure 2

```
// update Fields
function updateFields(A, N, PP) {
    document.getElementById("TotalMonths").innerHTML = N;
    document.getElementById("MonthlyPay").innerHTML = format_currency(PP,'£');
    document.getElementById("TotalPayment").innerHTML=
format_currency(PP*N,'£');
    document.getElementById("TotalInterest").innerHTML= format_currency((PP*N)-
A,'£'); }
```

Monthly Pay: £ 536.82	
Total of 360 Loan Payments	£ 193,255.78
Total Interest	£ 93,255.78

Figure 2: Summarized Loan Information

This function takes a decimal values as an argument and currency code and returns a string formatted value as shown in Monthly Pay Table below, [1]

```
// Format currency
function format_currency(n, currency) {
    return currency + " " + n.toFixed(2).replace(/./g, function(c, i, a) {
```

```
return i > 0 && c !== "." && (a.length - i) % 3 === 0 ? "," + c : c;
});
```

5.1 Monthly Schedule

The Monthly Amortization function takes parameters Loan Amount "**A**", TotalMonths "**N**", "MonthlyPay "**PP**" and Monthly Interest rate "**Rn**" and generate table for amortization schedule.

```
// Compute Monthly Amortization Schedule
function computeMonthly(A, N, Rn, PP) {
    var P_amount, interest, bal;
    var table = document.getElementById("alternatecolor");
    for (var i = 1; i < N+1; i++) {
        var rowCount = table.rows.length;
        var row = table.insertRow(rowCount);
        // Calculation of Principal amount, Interest nad Balance
        P_amount = PP/(Math.pow((1 + Rn), (1 + N - i)));
        interest = PP - P_amount;
        bal = (interest/Rn) - P_amount
        // Insert row in Amortization Table
        row.insertCell(1).innerHTML = format_currency(PP,'£');
        ...
        // Insert a row at the end of each Year
        if(i % 12 == 0){
            var row = table.insertRow();
            ... } // end IF
        } // end For
    altRows('alternatecolor'); // Make table rows in alternate colors }
```

The following schedule in Figure 3 is an output of the above code where **A**= "**100,000**", Duration **N=12** months and Interest Rate = **5%**. As it is evident that the interest rate is decreasing towards the end of the loan while the principal amount is increasing, on the other hand the payment mount is constant.

Monthly Schedule		Yearly Schedule		
		User's Choice		
Payment No.	Payment Amount	Principal Amount Paid	Interest Amount Paid	Loan Outstanding Balance
1	£ 8,560.75	£ 8,144.08	£ 416.67	£ 91,855.92
2	£ 8,560.75	£ 8,178.02	£ 382.73	£ 83,677.90
3	£ 8,560.75	£ 8,212.09	£ 348.66	£ 75,465.81
4	£ 8,560.75	£ 8,246.31	£ 314.44	£ 67,219.51
5	£ 8,560.75	£ 8,280.67	£ 280.08	£ 58,938.84
6	£ 8,560.75	£ 8,315.17	£ 245.58	£ 50,623.67
7	£ 8,560.75	£ 8,349.82	£ 210.93	£ 42,273.85
8	£ 8,560.75	£ 8,384.61	£ 176.14	£ 33,889.25
9	£ 8,560.75	£ 8,419.54	£ 141.21	£ 25,469.70
10	£ 8,560.75	£ 8,454.62	£ 106.12	£ 17,015.08
11	£ 8,560.75	£ 8,489.85	£ 70.90	£ 8,525.23
12	£ 8,560.75	£ 8,525.23	£ 35.52	£ -0.00
YEAR				

Figure 3: Monthly Schedule

5.2 Yearly Schedule

This is an extra feature that adds functionality to the amortization schedule to display yearly payments instead of monthly. The following is the handler in the JQuery for the "yearly_schedule" button click.

```
$('#yearly_schedule').click(function()
```

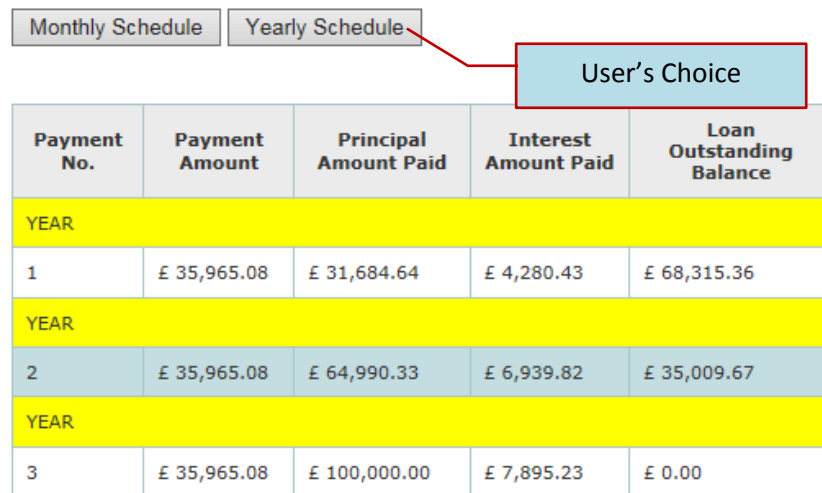
This function calls the same functions as monthly schedule one variable "N_yearly" is added which is defined as

```
N_Yearly = N * 12;
```

And instead of passing "N" to the methods "N_yearly" variable is passed as follows

```
updateFields(A,N_yearly,PP);
updateTable(A,N_yearly,Rn,PP);
altRows('alternatecolor');
updategraph(A,PP,N_yearly);
```

The following is the output of the yearly amortization schedule.



Payment No.	Payment Amount	Principal Amount Paid	Interest Amount Paid	Loan Outstanding Balance
YEAR				
1	£ 35,965.08	£ 31,684.64	£ 4,280.43	£ 68,315.36
YEAR				
2	£ 35,965.08	£ 64,990.33	£ 6,939.82	£ 35,009.67
YEAR				
3	£ 35,965.08	£ 100,000.00	£ 7,895.23	£ 0.00

Figure 4: Yearly Schedule

5.3 Constraints

Several constraint has been defined in the code at the input level to minimize the errors and overflows states during the execution of the code. This includes checking all the inputs from the users as defined in the following table.

Component Name	Constraints	Approved Values
Loan Amount	Decimal	1000 to 1 Billion
Loan Period in Months	Integer	1 to 1200 months
Loan Period in Years	Integer	1 to 99 Years
Interest Rate	Decimal	0 to 100

The following code is constraint on input amount

```
A = parseFloat($('input[name=amount]').val());
if(isNaN(A) || A<1000 || A>=10000000000) {
alert("The range is 1000 to 1 billion !"); }
```

5. INTERFACE DESIGN

The design is simple, I used Cascaded Style Sheet in external files to define font, color and spacing information to the web page. This includes defining of the "alternate rows" class. For the amortization schude to increase the readability of the table. The following interface for input the information.

Input Loan Information		
Loan Amount (£)	<input type="text" value="100000"/>	
Loan Period (Months)	<input type="text" value="360"/>	Months
Loan Period (Years)	<input type="text" value="30"/>	Years
Interest Rate	<input type="text" value="5"/>	%
<input type="button" value="Calculate"/>		

Figure 5: User Input Screen

The following is the code of CSS Style sheet used to define alternate rows for the amortization schedule.

Style.CSS [\[Download CSS file\]](#)

```
table.altrowstable {
    font-family: verdana,arial,sans-serif;
    font-size:11px;
    color:#333333;
    border-width: 1px;
    border-color: #a9c6c9;
    border-collapse: collapse; }
.oddrowcolor{
    background-color:#FFFFFF; }
.evenrowcolor{
    background-color:#c3dde0; }
```

Charts

To improve the design and adding interactivity, I used the Pie Chart from Highcharts which is based on pure JavaScript. The following JQuery Function used to display interactive line chart by passing **Rn**, **PP** and **N** input and inject to the Hichart API. [\[2\]](#)


```

function graph_IP(Rn,PP,N) {
var options;
  options = {
    series: [{

      data: (function () {
        // generate an array of data for N and Monthly Payments
        var data = [],
            time = N,
            i;
        for (i = 1; i <= N; i++) {
          P_amount = PP/(Math.pow((1 + Rn), (1 + N - i)));
          interest = PP - P_amount;
          data.push({
            x: i,
            y: interest
          });
        }
        return data;
      })()

    },

    step: 'right',
    name: 'Months'  ]
  };
  $('#container').highcharts(options); }

```

Four types of chart are being used to display Payment Breakdown using pie chart, and Interest, principal and Monthly Payment using Line chart. A user clicks on the buttons to change different graphs on the screen as Shown in following figures.

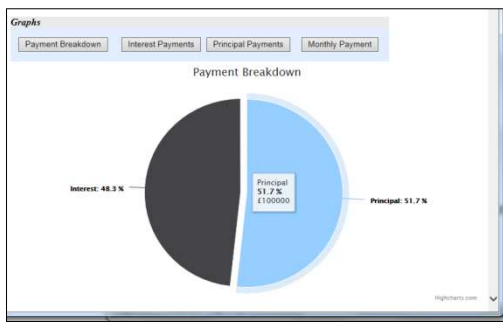
Graphs

Payment Breakdown

Interest Payments

Principal Payments

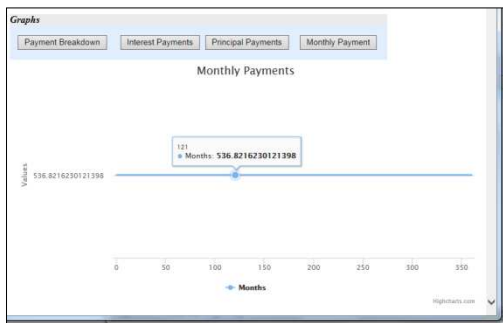
Monthly Payment



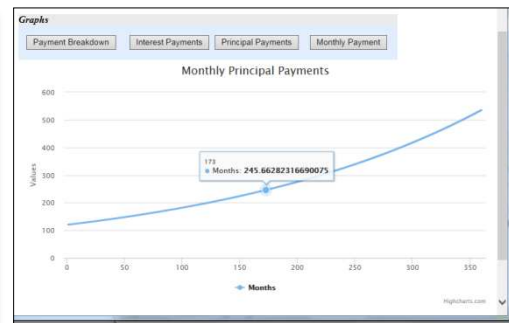
Payment BreakDown



Monthly Interest Payment



Monthly Payments



Principal Payments

1. References

- [1] A developers note, a Semi-technical web development blog, Javascript function to convert number to currency <https://adevelopersnotes.wordpress.com/tag/convert/> [Accessed March 2016]
- [2] Highcharts: Interactive JavaScript charts for your webpage, www.highcharts.com
- [3] Dr Georgios Koutsoukos, Lecture Notes on Financial Services Information Systems, University of Leicester