

# Regexps: Applications in Clinical Programming

Rohini Rao, Amgen, Seattle, WA  
Paul Hamilton, Amgen, Seattle, WA

## Abstract

Regular expressions are used by many programming languages to search for certain patterns in text. SAS is one such language that supports regular expressions for pattern matching. In the RX library of functions, RXPARSE is commonly used to parse a pattern in a string, be it a number, a letter, a character or a string of characters.

Often, clinical programmers are faced with the challenge of extracting numbers in a stream of alpha-numeric output. This paper focuses on one solution to this problem. In particular, it explains how to extract the matrix of coefficients from the Expected Mean Square output of a MIXED Procedure. This code is part of a macro to automate statistical processing in support of early drug development and discovery.

## Introducing RXPARSE

RXPARSE can be applied using default or user-defined character classes.

### Default Character Classes:

The user specifies a default character class with a dollar sign (\$) followed by a single upper- or lowercase letter. In the following list, the character class is listed in the left column and the definition is listed in the right column.

|               |                                                                                                      |
|---------------|------------------------------------------------------------------------------------------------------|
| \$a or<br>\$A | Matches any alphabetic upper- or lowercase letter in a substring ('\$a-zA-Z').                       |
| \$c or<br>\$C | Matches any character allowed in a version 6 SAS name that is found in a substring ('\$0-9a-zA-Z_'). |
| \$d or<br>\$D | Matches any digit in a substring ('\$0-9').                                                          |
| \$i or \$I    | Matches any initial character in a version 6 SAS name that is found in a substring ('\$a-zA-Z_').    |
| \$l or \$L    | Matches any lowercase letter in a substring ('\$a-z').                                               |
| \$u or<br>\$U | Matches any uppercase letter in a substring ('\$A-Z').                                               |

\$w or \$W Matches any white space character, such as blank, tab, backspace, carriage return, etc., in a substring.

The following examples illustrate the uses of default character classes.

| Pattern    | Input string | Position of match | Value of match |
|------------|--------------|-------------------|----------------|
| \$L or \$l | Y+3 STRIkES  | 9                 | k              |
| \$U or \$u | Y+3 STRIkES  | 1                 | Y              |
| \$D or \$d | Y+3 STRIkES  | 3                 | 3              |

### User-defined Character Classes

A user-defined character class begins with a dollar sign (\$) and is followed by a string in quotation marks. A character class matches any one character within the quotation marks.

The following examples illustrate the uses of user-defined character classes.

| Pattern   | Input string | Position of match | Value of match |
|-----------|--------------|-------------------|----------------|
| \$'abcde' | 3+yE strikes | 11                | e              |
| \$'1-9'   | 3+yE strikes | 1                 | 3              |

### Application of RXPARSE

The Expected Mean Square (EMS) matrix below is part of the output from a MIXED Procedure.

1.15 Var(Residual) + 7.72 Var(FieldA(FieldB)) + 25 Var(FieldB)

3.13 Var(Residual) + 5.76 Var(FieldA(FieldB))

2.77 Var(Residual)

The goal is to extract the coefficients from the EMS matrix such that we have the following:

1.15      7.72      25

3.13      5.76

2.77

The code below, which is part of a larger macro, demonstrates one method of parsing the coefficients. The macro has five parameters but the one that is of interest to us is 'type1'. It is the dataset that contains the EMS variable.

We first count the maximum number of coefficients in the string by looking for the "Var(" expression that appears after each coefficient. We then create a macro variable, nCoeff, to denote this number.

```
%macro getCoeff2 (type1 =, SolutionF =, CovParms =, prob = 0.95, mode = normal);
data &type1B;
  set type1;
  ems = compbl(ems);    * removes multiple blanks between words;
  ems = left(ems);
  attrib terms
  length = 8
  format = 2.
  label = '# Terms';
  terms = count(ems, 'Var()');
run;
proc print; var terms; run;

proc sql noprint;
  select max(terms) into :nCoeff
  from &type1B;
quit;
%put &nCoeff.;
%let nCoeff=%sysfunc(compress(&nCoeff));
%put &nCoeff;
```

The next piece of code looks at the entire string and extracts the first coefficient. It then creates a substring such that the second coefficient is now at the beginning of the string and is extracted. This process is repeated until all the coefficients have been extracted.

We are using the first string in the EMS matrix, "1.15 Var(Residual) + 7.72 Var(FieldA(FieldB)) + 25 Var(FieldB)" to better explain the code.

We first parse the entire string for the first digit of the first coefficient, which in our example is 1 in 1.15. Next, we match its position, which, in this case equals 1. Then, we locate the first space after the set of characters in the coefficient. Here, we are looking for the space after 1.15. We make the assumption that the coefficient will not be more than ten characters long. Finally, we extract all the coefficient characters before the space. In this case, match1=1 and space1-1=4, so substr(ems1,match1,space1-1)=1.15.

```

data &type1.1;
set &type1.B;
* Create ems1 and sublen1 to fit the array below ;
ems1=ems;
sublen1=length(ems);
rx1=rxparse("$d");           *parse the first digit of the first coefficient;
match1=rxmatch(rx1,ems1);    *match the first digit;
if match1 ne 0 then do;
    space1=anyospace(substr(ems1,match1,10)); *the first space after the coefficient;
    coeff1=substr(ems1,match1,space1-1);      *pull all the digits before the space;
end;

```

We repeat the above process for the remaining coefficients. In our example, the maximum number of coefficients is 3. 1.15 has been extracted and 7.72 and 25 remain to be extracted.

We first determine the length of the substring that begins after the previous coefficient. Here, the substring starts after 1.15.

For i=2, sublen2=sublen1 - (match1 + space1) or  
sublen2 =length(ems)-(1 + 5).

The new ems string starts at the end of the previous coefficient.

For i=2, ems2 = substr(ems1,match1 + space1, sublen2) or  
ems2 = substr(ems, 1 + 5, sublen2).

We parse the new ems string for the first digit of the first of the remaining coefficients. In our example, 7.72 and 25 remain to be extracted. For i=2, rx2 will identify the first digit in 7.72. We next match the position of this digit and then locate the first space after the coefficient, which in this case will be the space after 7.72. Finally, as before, we use this space to extract all the coefficient characters before it.

```

array emsCnt(*) ems1-ems&nCoeff;
array sublenCnt(*) sublen1-subLen&nCoeff;
array rxCnt(*) rx1-rx&nCoeff;
array matchCnt(*) match1-match&nCoeff;
array spaceCnt(*) space1-space&nCoeff;
array coeffCnt(*) coeff1-coeff&nCoeff;

do i=2 to &nCoeff;
    * The new EMS string starts at the end of the previous coefficient ;
    if matchCnt{i-1} ne 0 then do;
        * For i=2, sublen2=length(ems)-(1 + 5);
        sublenCnt{i}=sublenCnt{i-1}-(matchCnt{i-1}+spaceCnt{i-1});
        * For i=2, ems2 = substr(ems1, 1 + 5, sublen2). Note that ems1=ems;
        emsCnt{i}=substr(emsCnt{i-1},matchCnt{i-1}+spaceCnt{i-1},sublenCnt{i});
    end;
end;

```

```

* For i=2, parse the first 7 in 7.72;
rxCnt{i}=rxparse("$d");

* For i=2, match the position of the first 7 in 7.72 ;
matchCnt{i}=rxmatch(rxCnt{i},emsCnt{i});

*For i=2, use the first space after 7.72 to extract the coefficient.
if matchCnt{i} ne 0 then do;
    spaceCnt{i}=anyspace(substr(emsCnt{i},matchCnt{i},10));
    coeffCnt{i}=substr(emsCnt{i},matchCnt{i},spaceCnt{i}-1);
end;
end;
run;
%mend;

```

The array is repeated until all the coefficients are extracted. The final step for this portion of the macro is to convert the coefficients into numerical variables and feed them to an IML table.

## Conclusion

Previous attempts to extract the matrix of coefficients were made by using traditional character functions to eliminate all the characters that were not coefficients. However, this approach proved to be very difficult because of the nature of the output that we are parsing. There is no consistent pattern with the characters that are not coefficients. As well, traditional character functions treat <( )> as word delimiters. For instance, the phrase “Var(“ appears after each coefficient and is therefore used to obtain the maximum number of coefficients in the string. However functions such as TRANWRD( ) or SCAN( ) render the phrase useless by treating <( )> as a delimiter. With RXPARSE, we are able to extract the coefficients regardless of what the remaining characters look like, making it to be a more global approach.

Regular expressions have been around for a long time. They have proven to be powerful across several languages. This paper highlights the usefulness of RXPARSE in the context of a SAS Data step in a clinical programming application.

## **Contact Information**

Rohini Rao  
[rorao@yahoo.com](mailto:rorao@yahoo.com)

Paul Hamilton  
[hamilton@amgen.com](mailto:hamilton@amgen.com)

## **Acknowledgements**

SAS is a registered trademark of SAS Institute, Inc. in the U.S.A and other countries. Other brand and product names are registered trademarks or trademarks of their respective companies.