

# POLYc and utilities to extract numeric coefficients of symbolic polynomials (HP 50g)

D.A. Burkett (0db)

20 Mar 2026

## Contents

|     |                                                                   |   |
|-----|-------------------------------------------------------------------|---|
| 1   | Introduction                                                      | 1 |
| 2   | Summary of Programs                                               | 2 |
| 3   | Using POLYc                                                       | 2 |
| 4   | Utility Programs                                                  | 4 |
| 4.1 | POLYr: Coefficients of a rational polynomial . . . . .            | 5 |
| 4.2 | PCHER: Hermite polynomial coefficients . . . . .                  | 5 |
| 4.3 | PCLEG: Legendre polynomial coefficients . . . . .                 | 5 |
| 4.4 | PCCHE: Chebyshev polynomial coefficients . . . . .                | 6 |
| 4.5 | PCLAG: Lagrange polynomial coefficients . . . . .                 | 6 |
| 4.6 | PCCYC: Cyclotomic polynomial coefficients . . . . .               | 6 |
| 4.7 | PCTAY: Taylor polynomial coefficients of a polynomial . . . . .   | 7 |
| 4.8 | PCRAT: Taylor polynomial coefficients for rational polynomial . . | 7 |
| A   | Source Code                                                       | 8 |

## 1: Introduction

The User RPL program POLYc extracts the numeric coefficients of a polynomial in one variable, where the polynomial is an algebraic expression such as  $X^4 + 3X^2 - 7X + 5$ . The coefficients are returned as a vector compatible with PEVAL (evaluate a polynomial) and PROOT (find polynomial roots). Native<sup>1</sup> functions such as HERMITE, LEGENDRE, and others return symbolic polynomials and POLYc can be used to get the coefficients of those polynomials.

POLYc is intended to be used in Numeric Approx mode,<sup>2</sup> not EXACT mode. Any undefined variable can be used as the polynomial variable. The coefficients must be explicit numeric constants, not variables, but they may be real or complex numbers. The polynomial need not be simplified to the canonical sum-of-terms form.

Several utility programs are included to return polynomial coefficients resulting from native polynomial commands and to find the coefficients of a rational polynomial.

Although POLYc and the utility programs implement some simple error trapping, it is possible that intermediate results may be left on the stack if errors occur. The utility programs call native functions such as LAGRANGE

---

<sup>1</sup>‘Native’ means a ‘built-in’ function or command, as opposed to a user program.

<sup>2</sup>‘Numeric Approx’ means that Numeric and Approx are checked in the CAS (computer algebra system) mode menu. Exact mode means that they are unchecked.

and **LEGENDRE** which can leave input arguments on the stack after errors. Some of those native functions require **EXACT** mode so **PUSH** and **POP** are used to save and restore the system flags. The programs are designed so that a **PUSH** is followed by a **POP** whether or not errors are raised. Depending on the error, though, it is possible that the **POP** is not executed.

The algorithm used by **POLYc** is specified in the code description (p.8). Several **CAS** function calls are used for each coefficient, so **POLYc** is slow. This is not a program you want to execute hundreds of times as part of another program.

The **POLYc** functionality is regrettably unavailable as a native command. If the internal structure of a polynomial in the **CAS** was known, it would certainly be faster to step through the structure and extract the coefficients.

Since **POLYc** uses several **CAS** commands but runs in **Numeric Approx** mode, it is necessary to configure some flags and use some special coding methods to prevent the **CAS** from complaining and raising prompts while **POLYc** executes. For that reason **POLYc** may be an instructive example of writing code with **CAS** functions in **Numeric Approx** mode.

Incidentally, the inverse operation of creating a symbolic polynomial from a list of coefficients is trivial with **PEVAL**:

$$[ \ 3 \ 2 \ 1 \ ] \ 'X' \ \text{PEVAL} \Rightarrow '1.+(2.+3.*X)*X'$$

**EXPAND** can be used to put the polynomial into sum-of-terms form, but **CAS** flags will probably have to be adjusted.

The source code for the programs are text files using tri-graph characters, which can be downloaded to the calculator. The file name format is *name.hp*.

## 2: Summary of Programs

This section summarizes the programs included with this package. Command names in parentheses specify the native ('built-in') function which provides the symbolic polynomial.

| Program Name | Description                                                               |
|--------------|---------------------------------------------------------------------------|
| <b>POLYc</b> | Extract symbolic polynomial numerical coefficients as a vector            |
| <b>POLYr</b> | Extract coefficients of a rational polynomial                             |
| <b>PCHER</b> | Find coefficients of an Hermite polynomial ( <b>HERMITE</b> )             |
| <b>PCLEG</b> | Find coefficients of a Legendre polynomial ( <b>LEGENDRE</b> )            |
| <b>PCCHE</b> | Find coefficients of a Chebyshev polynomial ( <b>TCHEBYCHEFF</b> )        |
| <b>PCLAG</b> | Find coefficients of a Lagrange polynomial ( <b>LAGRANGE</b> )            |
| <b>PCCYC</b> | Find coefficients of a cyclotomic polynomial ( <b>CYCLOTOMIC</b> )        |
| <b>PCTAY</b> | Coefficients of a Taylor polynomial of a polynomial ( <b>PTAYL</b> )      |
| <b>PCRAT</b> | Coefficients of Taylor polynomial of rational polynomial ( <b>DIVPC</b> ) |

## 3: Using POLYc

The stack I/O for **POLYc** is

$$1 \mid \underline{ 'p(x)' } \mid \Rightarrow \mid \underline{ [ a_i ] } \mid$$

where ' $p(x)$ ' is the polynomial as an algebraic expression in some variable  $x$  and  $[a_i]$  is the coefficient vector. For

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0 \quad (3.1)$$

the coefficients are  $[a_i] = [a_n, a_{n-1}, \dots, a_1, a_0]$ . For example,

$$'3.*X^2. + 2.*X - 1.' \text{ POLYc} \Rightarrow [3. \ 2. \ -1.]$$

The following constraints apply.

- $p(x)$  may be an explicit polynomial in only one variable, a global variable name, or a numeric constant; see the examples below.
- $p(x)$  must actually *be* a polynomial, and not some other type of expression. POLYc does not verify that  $p(x)$  is a polynomial; if it is not, unpredictable results may occur which are not trapped by error handling.
- $p(x)$  need not be in the canonical, sum-of-terms form of equation (3.1) above. Any expression that simplifies to a polynomial in one variable can be used.
- $x$  may be any variable name, but the name must refer to a global variable which is undefined in the current directory path. One safe policy is to use the default CAS variable 'X' and leave it undefined. POLYc identifies the variable from the polynomial.
- The coefficients  $a_i$  must be explicit constants, and may be real or complex numbers.

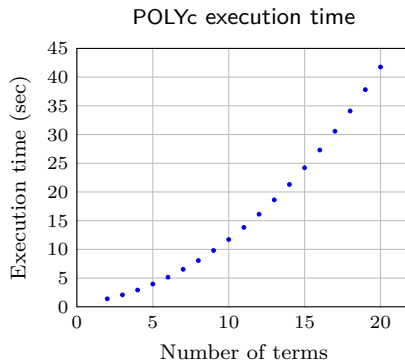
POLYc may raise the following error messages.

| Error Message             | Description                                                                                                           |
|---------------------------|-----------------------------------------------------------------------------------------------------------------------|
| POLYc: No input argument  | One argument must be on stack level 1                                                                                 |
| POLYc: Invalid input type | $p(x)$ must be an algebraic object, a global variable name, or a numeric constant.                                    |
| POLYc: # vars $\neq$ 1    | The polynomial must have only one variable.                                                                           |
| POLYc: <i>message</i>     | Some other error occurred. <i>message</i> is the string for the error. Intermediate results may be left on the stack. |

Some examples are shown below. Examples 1) and 2) show typical usage with polynomials in canonical form with real coefficients. 3) and 4) show the extraction of complex coefficients, either in the form  $(a + b * i)$  or  $(a, b)$ . Example 5) shows that symbolic constants are converted to numerical values. Example 6) shows the unusual case where the polynomial has an explicit leading coefficient of zero. Example 7) shows that the polynomial need not be in canonical form. If the input argument is a single global variable, as shown in example 8), the vector  $[1 \ 0]$  is returned. The single global variable is interpreted as a first-degree polynomial with no constant term. If the input argument is a numeric constant, real or complex, it is returned as a single-element vector. The input argument is interpreted as a zero-degree polynomial, with only a constant term.

|    | $p(x)$                                               | POLYc Result                 |
|----|------------------------------------------------------|------------------------------|
| 1) | '4*X <sup>3</sup> - 2*X <sup>2</sup> + X - 5'        | [ 4. -2. 1. -5. ]            |
| 2) | 'X <sup>5</sup> '                                    | [ 1. 0. 0. 0. 0. 0. ]        |
| 3) | '(3 + 4*i)*X <sup>2</sup> - (5 - 6*i)*X + (7 + 8*i)' | [ (3.,4.) (-5.,6.) (7.,8.) ] |
| 4) | '(3.,4.)*X <sup>2</sup> - (5.,6.)*X + (7.,8.)'       | [ (3.,4.) (-5.,6.) (7.,8.) ] |
| 5) | ' $\pi^2 X^2 + \pi X - \pi$ '                        | [ 9.869 3.141 -3.141 ]       |
| 6) | '0*X + 2'                                            | [ 0. 2. ]                    |
| 7) | 'X*(X-5)*(3*X + 2) + 7'                              | [ 3. -13. -10. 7. ]          |
| 8) | 'X'                                                  | [ 1. 0. ]                    |
| 9) | 3.7                                                  | [ 3.7 ]                      |

POLYc is slow. The plot below shows the execution time as function of the number of polynomial terms. The number of terms includes the constant term, so a 2-term polynomial would be  $ax + b$ .



It would take about 5.5 minutes to find the coefficient vectors for all 19 of the test polynomials which range from 2 to 20 terms.

#### 4: Utility Programs

This section describes utility programs which use POLYc. POLYr finds the coefficient vectors of a rational polynomial. Other programs find the coefficients of special polynomials returned by native functions. These functions use the current CAS variable ('X', by default).

All these programs call POLYc, so the input argument constraints for POLYc apply.

#### 4.1 POLYr: Coefficients of a rational polynomial

$$\begin{array}{c|c} 2 & \\ \hline 1 & 'p(x)/q(x)' \end{array} \Rightarrow \begin{array}{c|c} & [p_i] \\ \hline & [q_i] \end{array}$$

POLYr returns the coefficient vectors  $[p_i]$  and  $[q_i]$ , where  $p_i$  are the coefficients of the numerator polynomial and  $q_i$  are the coefficients of the denominator polynomial. POLYr calls POLYc with  $p(x)$  and  $q(x)$  so the constraints for POLYc apply. ' $p(x)/q(x)$ ' must be an explicit rational polynomial, not a variable name. The variable of  $p(x)$  need not be the same as the variable of  $q(x)$ , but both variables must be undefined in the current directory path.

Example: ' $(6*X^2 + 5*X + 4) / (3*X^2 + 2*X)$ ' POLYr  $\Rightarrow$  [6. 5. 4.] [3. 2. 0.]

| POLYr Error Messages                                 |                                         |
|------------------------------------------------------|-----------------------------------------|
| POLYr: No input arg                                  | Must be at least one argument on stack  |
| POLYr: Invalid input type                            | $p(x)/q(x)$ must be an algebraic object |
| Other error messages may be raised by FXND or POLYc. |                                         |

#### 4.2 PCHER: Hermite polynomial coefficients

$$1 \begin{array}{c|c} & n \end{array} \Rightarrow \begin{array}{c|c} & [a_i] \end{array}$$

PCHER calls the native HERMITE function to return the coefficients  $[a_i]$  of the Hermite polynomial of order  $n$ .  $n$  must be a non-negative integer.

Example: 3 PCHER  $\Rightarrow$  [ 8. 0. -12. 0. ]

| PCHER Error Messages                                    |                                        |
|---------------------------------------------------------|----------------------------------------|
| PCHER: No input arg                                     | Must be at least one argument on stack |
| PCHER: Invalid input type                               | $n$ must be an explicit real number    |
| Other error messages may be raised by HERMITE or POLYc. |                                        |

#### 4.3 PCLEG: Legendre polynomial coefficients

$$1 \begin{array}{c|c} & n \end{array} \Rightarrow \begin{array}{c|c} & [a_i] \end{array}$$

PCLEG calls the native LEGENDRE function to return the coefficients  $[a_i]$  of the Legendre polynomial of order  $n$ .  $n$  must be a non-negative integer.

Example: 3 PCLEG  $\Rightarrow$  [ 2.5 0. -1.5 0. ]

| PCLEG Error Messages                                     |                                        |
|----------------------------------------------------------|----------------------------------------|
| PCLEG: No input arg                                      | Must be at least one argument on stack |
| PCLEG: Invalid input type                                | $n$ must be an explicit real number    |
| Other error messages may be raised by LEGENDRE or POLYc. |                                        |

#### 4.4 PCCHE: Chebyshev polynomial coefficients

$$1 \mid \boxed{n} \Rightarrow \mid \boxed{[a_i]}$$

PCCHE calls the native TCHEBYCHEFF<sup>1</sup> function to return the coefficients  $[a_i]$  of the Chebyshev polynomial of order  $n$ .  $n$  must be a non-negative integer.

Example: 3 PCCHE  $\Rightarrow$  [ 4. 0. -3. 0. ]

| PCCHE Error Messages                                        |                                        |
|-------------------------------------------------------------|----------------------------------------|
| PCCHE: No input arg                                         | Must be at least one argument on stack |
| PCCHE: Invalid input type                                   | $n$ must be an explicit real number    |
| Other error messages may be raised by TCHEBYCHEFF or POLYc. |                                        |

#### 4.5 PCLAG: Lagrange polynomial coefficients

$$1 \mid \boxed{[xy]} \Rightarrow \mid \boxed{[a_i]}$$

PCLAG calls the native LAGRANGE function to return the coefficients  $[a_i]$  of the interpolating polynomial for data points  $[xy]$ .  $[xy]$  must be an explicit matrix, not a ticked name, and the elements of  $[xy]$  must be real numbers. The set of points  $[xy]$  is specified as the matrix

$$\begin{bmatrix} x_1 & x_2 & \dots & x_n \\ y_1 & y_2 & \dots & y_n \end{bmatrix}$$

Example: [ [ 1 3 4 ] [ 3 29 54 ] ] PCLAG  $\Rightarrow$  [ 4 -3 2 ]

| PCLAG Error Messages                                     |                                        |
|----------------------------------------------------------|----------------------------------------|
| PCLAG: No input arg                                      | Must be at least one argument on stack |
| PCLAG: Invalid input type                                | $[xy]$ must be an explicit real matrix |
| Other error messages may be raised by LAGRANGE or POLYc. |                                        |

#### 4.6 PCCYC: Cyclotomic polynomial coefficients

$$1 \mid \boxed{n} \Rightarrow \mid \boxed{[a_i]}$$

PCCYC calls the native CYCLOTOMIC function to return the coefficients  $[a_i]$  of the cyclotomic polynomial of order  $n$ .  $n$  must be a non-negative integer.

Example: 10 PCCYC  $\Rightarrow$  [ 1 -1 1 -1 1 ]

| PCCYC Error Messages                                       |                                            |
|------------------------------------------------------------|--------------------------------------------|
| PCCYC: No input arg                                        | Must be at least one argument on stack     |
| PCCYC: Invalid input type                                  | $n$ must be an explicit real integer $> 0$ |
| Other error messages may be raised by CYCLOTOMIC or POLYc. |                                            |

<sup>1</sup>There are at least six different ways to spell this Russian name in English. I just prefer Chebyshev.

## 4.7 PCTAY: Taylor polynomial coefficients of a polynomial

$$\begin{array}{c|c} 2 & P(x) \\ \hline 1 & x_0 \end{array} \Rightarrow \begin{array}{c|c} & \\ \hline & [a_i] \end{array}$$

PCTAY returns the coefficients  $[a_i]$  of the Taylor polynomial at  $x = x_0$  of polynomial  $P(x)$ .  $P(x)$  must be an explicit polynomial and  $x_0$  must be an explicit real number. PCTAY uses the native function PTAYL to obtain the symbolic polynomial.

Example: Given  $P(x) = x^2 + 3x + 2$ , find the coefficients of the Taylor polynomial at  $x_0 = 3.6$ : 'X^2 + 3\*X + 2' 3.6 PCTAY  $\Rightarrow$  [1. 10.2 25.76]

| PCTAY Error Messages                                  |                                                                               |
|-------------------------------------------------------|-------------------------------------------------------------------------------|
| PCTAY: <2 input args                                  | Must be at least two arguments on the stack                                   |
| PCTAY: Invalid input arg(s)                           | $x_0$ must be an explicit real number and $P(x)$ must be an algebraic object. |
| Other error messages may be raised by PTAYL or POLYc. |                                                                               |

## 4.8 PCRAT: Taylor polynomial coefficients for rational polynomial

$$\begin{array}{c|c} 3 & N(x) \\ \hline 2 & D(x) \\ \hline 1 & d \end{array} \Rightarrow \begin{array}{c|c} & \\ \hline & [a_i] \end{array}$$

PCRAT returns the coefficients  $[a_i]$  of the Taylor polynomial of degree  $d$  for the rational polynomial  $N(x)/D(x)$ .  $N(x)$  and  $D(x)$  must be explicit polynomials and  $d$  must be an explicit real number integer. PCRAT uses the native DIVPC function to obtain the symbolic polynomial.

Example: To find the coefficients of the Taylor polynomial of degree 4 for the rational polynomial

$$\frac{x^3 + 4x + 12}{x^{11} + 2}$$

use: 'X^3 + 4\*X + 12' 'X^11 + 2' 4 PCRAT  $\Rightarrow$  [ .5 0. 2. 6. ]

| PCRAT Error Messages                                  |                                                                                           |
|-------------------------------------------------------|-------------------------------------------------------------------------------------------|
| PCRAT: <3 input args                                  | Must be at least three input arguments                                                    |
| PCRAT: Invalid input arg(s)                           | $d$ must be an explicit real number integer, $N(x)$ and $P(x)$ must be algebraic objects. |
| Other error messages may be raised by DIVPC or POLYc. |                                                                                           |

## A: Source Code

This section lists the User RPL source code for POLYc and the other utility programs described in this note.

### POLYc Code

The POLYc source code is described in detail in this section.<sup>1</sup> The table below outlines the algorithm.<sup>2</sup>

|                                                                                                                       |                 |
|-----------------------------------------------------------------------------------------------------------------------|-----------------|
| The original polynomial, in variable $x$                                                                              | $3x^2 + 2x + 5$ |
| Set $x = 0$ then evaluate the polynomial, which gives the coefficient, and add it to the list of coefficients $[a_i]$ | 5               |
| Subtract the coefficient (5) from the polynomial                                                                      | $3x^2 + 2x$     |
| Divide the result by $x$ and simplify                                                                                 | $3x + 2$        |
| Repeat from the second step above until the highest-degree term is done.                                              | $[a_i]$         |

CAS functions are used to manipulate the polynomial but we want to run the program in **Numeric Approx** mode. This has consequences. Many CAS functions require system flag `-3 clear` (return symbolic results) and system flag `-105 clear` (exact calculations). Some commands need **Complex** mode enabled (`-103 set`), but we need that, anyway, for complex coefficients. Other commands require **RADIANT** angle mode.<sup>3</sup> The CAS raises errors or prompts if any of these settings are incorrect.<sup>4</sup> Since POLYc changes so many settings, **PUSH** and **POP** are used to save the flags before they are changed and to restore the flags when POLYc terminates. POLYc traps errors with **DOERR** so each error handler must also use **POP** to restore the flags.

The coefficients may well be floating-point numbers and some CAS commands object to this, regardless of the flag and mode settings. This is avoided by using `→Q` to convert the coefficients to rational fractions. `→Q` uses the current number format (**STD**, **FIX**, etc.) to set the precision of the result, so POLYc sets **STD** mode for full 12-digit precision. `→NUM` converts the rational fractions to floating-point, after all the coefficients are found and the coefficient list is converted into a vector.

Any fixed numeric constants used with CAS commands should be exact integers or rational fractions, and this adds another complication. Suppose POLYc is saved on the calculator but we need to edit it. If the file is opened for editing in **Numeric Approx** mode, *all* of the exact integers will be converted to floating-point numbers with decimal points. The CAS will object the next time POLYc is executed. This also happens if the file is sent to the emulator or a physical calculator in **Numeric Approx** mode. This can be prevented in several ways, but POLYc uses the method I call ‘quoted code’.<sup>5</sup> All of the sensitive CAS code is entered as a text string, delimited with double quotes (") as any other string. Since the numeric constants are ‘hidden’ in

<sup>1</sup>This is the first 50g program I’ve written using CAS commands; I use Mathematica for CAS-type stuff. So you’re really taking your chances with this one.

<sup>2</sup>This is a simple-minded approach because I’m a simple-minded person.

<sup>3</sup>Although it may not be obvious just *why* this is so.

<sup>4</sup>Which is quite vexing.

<sup>5</sup>Which I did not invent or discover, but saw in another programmer’s code.

the string, they are not changed and remain exact numbers. The quoted code is executed with `OBJ→`. If a string is used in the quoted code, the double quotes for the string must be prefixed with the `\` escape character.

Some other points of interest:

- It is necessary in some cases, depending on the form of the input polynomial, to use `EVAL` and `SIMPLIFY` after some CAS operations so that subsequent functions operate correctly.
- `LNAME` is used both to get the polynomial variable and to verify that the polynomial has only one variable. `LNAME` works in `Numeric Approx` mode, but is included in the quoted code, anyway.
- The `DEGREE` function finds the degree of the polynomial, which is used as the `START` loop limit. It would be interesting to determine if `DEGREE` could also be used to verify that an expression actually is a polynomial. According to the AUR, the operation of `DEGREE` is a little byzantine, depending on the number of variables and their location in the current directory path. `DEGREE` returns 0 for a constant expression and  $-1$  if the expression is zero, but the AUR doesn't specify the result if the expression is not really a polynomial.
- The polynomial is evaluated for  $x = 0$  with the 'where' operator `|` and the replacement specification list `{v 0}`. `→LIST` is used to make this list, but I've always thought there should be a more simple method to make a list with variable name elements, directly with `{v 0}`. This works outside of `POLYc`, in a simple test program, but causes `POLYc` to fail. Why?
- If the input argument is a numeric constant, real or complex, it is returned as a single-element vector. This is consistent with the notion that a numeric constant can be considered as a polynomial of degree zero. This behavior also gives correct results with `POLYr` when the numerator or denominator is a numeric constant.
- If the input argument is a single global variable name, the vector `[ 1. 0. ]` is returned. This represents a first-degree polynomial with no constant term.
- `POLYc` uses the CAS commands in a very simple way and does not actually do any significant calculations. More sophisticated applications may require configuring additional CAS flags for proper operation. In fact, `POLYc` may depend on some CAS flag settings of which I am unaware, but happen to be set to the correct default values.

```

«
Define local variables. v is the polynomial variable, c is the coefficient list,
and t is the object type of the input argument.
'X' {} 0. → v c t
«
Save system flags
PUSH
Set flags and modes for CAS compatibility,
set STD display mode for full accuracy with →Q.
-3. CF -105. CF -103. SF RAD STD
CASE
Verify one stack input argument
DEPTH 1. < THEN POP "POLYc: No input argument" DOERR END
If input is a numeric constant then return it as a one-element vector
DUP TYPE 't' ► 0. == t 28. == t 1 == OR OR THEN POP 1. →ARRY END
If input is a single global variable then return a 2-element vector
t 6. == THEN POP DROP [1. 0.] END
Extract the coefficients of a symbolic polynomial
t 9. == THEN
Clear previous errors and begin error trapping
ERR0 IFERR
Begin quoted code
"

Verify that the polynomial has only one variable, else raise error
LNAME DUP SIZE EVAL
1 ≠ « POP DROP2 \"# vars ≠ 1\" DOERR » IFT
Save the polynomial variable name
1 GET 'v' STO
Convert floating-point coefficients to rational numbers
→Q DUP
Find the polynomial degree and start the coefficient extraction loop
DEGREE 1 SWAP START
Evaluate the polynomial with x = 0 and add the coefficient to list
DUP v 0 2 →LIST | EVAL DUP 'c' STO+
Subtract the coefficient from the polynomial
- EVAL
Divide the polynomial by the variable and simplify
v / SIMPLIFY
End the coefficient extraction loop
NEXT
Add the last coefficient to the list
'c' STO+
Convert the coefficient list to a vector;
convert the coefficients to floating-point numbers
c AXL →NUM
End quoted code
"

```

```

Evaluate the quoted code
OBJ→
Handle trapped error, if any
THEN
Restore system flags and raise error with the system error message
POP "POLYc: " ERRM + DOERR
End error trapping
END
Restore system flags
POP
END
Raise error for invalid input type
POP DROP "POLYc: Invalid input type" DOERR
END
» »

```

---

### POLYr Code – Coefficients of rational polynomial

POLYr finds the two coefficient vectors for a rational polynomial  $N(x)/D(x)$ . The native function FXND is used to get the symbolic polynomials  $N(x)$  and  $D(x)$ . CAS system flags must be set and restored. POLYr is particularly slow because POLYc is called twice.

|       |                                |
|-------|--------------------------------|
| POLYr | (244.5 bytes, checksum #2CE8h) |
|-------|--------------------------------|

```

«
Save system flags and set CAS flags
PUSH
-3 CF -105 CF -103 SF RAD STD
Verify stack argument and validate argument type
DEPTH 1. < « POP "POLYr: No input argument" DOERR » IFT
DUP TYPE 9. ≠ « POP DROP "POLYr: Invalid input type" DOERR » IFT
Clear previous errors
ERR0
Get numerator and denominator symbolic polynomials,
with error trapping; find coefficient vectors for each polynomial.
IFERR FXND SWAP POLYc SWAP POLYc
Handle error if raised.
THEN POP ERRM DOERR
END
Restore system flags
POP
»

```

---

**PCCHE Code – Coefficients of Chebyshev polynomial**

PCCHE returns the polynomial coefficient vector for the Chebyshev polynomial of order  $n$ . The native TCHEBYCHEFF command is used, which runs without errors in Numeric Approx mode so the system flags need not be saved and restored.

| PCCHE                                                   | (177 bytes, checksum #CB55h) |
|---------------------------------------------------------|------------------------------|
| «                                                       |                              |
| <i>Verify stack argument and validate argument type</i> |                              |
| DEPTH 0. == « "PCCHE: No input arg" DOERR » IFT         |                              |
| DUP TYPE 0. ≠ « "PCCHE: Invalid input type" DOERR » IFT |                              |
| <i>Clear previous errors</i>                            |                              |
| ERR0                                                    |                              |
| <i>Get polynomial, with error trapping</i>              |                              |
| IFERR TCHEBYCHEFF                                       |                              |
| <i>Handle error</i>                                     |                              |
| THEN "PCCHE: " ERRM + DOERR                             |                              |
| <i>No error; find polynomial coefficients</i>           |                              |
| ELSE POLYc                                              |                              |
| END                                                     |                              |
| »                                                       |                              |

**PCCYC Code – Coefficients of cyclotomic polynomial**

PCCYC returns the polynomial coefficient vector for the cyclotomic polynomial of order  $n$ . The native CYCLOTOMIC command is used, which runs without errors in Numeric Approx mode so the system flags need not be saved and restored.

| PCCYC                                                   | (177 bytes, checksum #CA34h) |
|---------------------------------------------------------|------------------------------|
| «                                                       |                              |
| <i>Verify stack argument and validate argument type</i> |                              |
| DEPTH 0. == « "PCCYC: No input arg" DOERR » IFT         |                              |
| DUP TYPE 0. ≠ « "PCCYC: Invalid input type" DOERR » IFT |                              |
| <i>Clear previous errors</i>                            |                              |
| ERR0                                                    |                              |
| <i>Get polynomial, with error trapping</i>              |                              |
| IFERR CYCLOTOMIC                                        |                              |
| <i>Handle error</i>                                     |                              |
| THEN "PCCYC: " ERRM + DOERR                             |                              |
| <i>No error; find polynomial coefficients</i>           |                              |
| ELSE POLYc                                              |                              |
| END                                                     |                              |
| »                                                       |                              |

## PCHER Code – Coefficients of Hermite polynomial

PCHER returns the polynomial coefficient vector for the Hermite polynomial of order  $n$ . The native HERMITE command is used, which runs without errors in Numeric Approx mode so the system flags need not be saved and restored.

| PCHER                                                   | (177 bytes, checksum #A8CEh) |
|---------------------------------------------------------|------------------------------|
| «                                                       |                              |
| <i>Verify stack argument and validate argument type</i> |                              |
| DEPTH 0. == « "PCHER: No input arg" DOERR » IFT         |                              |
| DUP TYPE 0. ≠ « "PCHER: Invalid input type" DOERR » IFT |                              |
| <i>Clear previous errors</i>                            |                              |
| ERR0                                                    |                              |
| <i>Get polynomial, with error trapping</i>              |                              |
| IFERR HERMITE                                           |                              |
| <i>Handle error</i>                                     |                              |
| THEN "PCHER: " ERRM + DOERR                             |                              |
| <i>No error; find polynomial coefficients</i>           |                              |
| ELSE POLYc                                              |                              |
| END                                                     |                              |
| »                                                       |                              |

## PCLAG Code – Coefficients of Lagrange polynomial

PCLAG returns the coefficient vector for a Lagrange interpolating polynomial. The native function LAGRANGE is used to find the symbolic polynomial. CAS system flags must be set and restored since LAGRANGE fails with CAS prompts in Numeric Approx mode.

| PCLAG                                                   | (250.5 bytes, checksum #BC4Dh) |
|---------------------------------------------------------|--------------------------------|
| «                                                       |                                |
| <i>Verify stack argument and validate argument type</i> |                                |
| DEPTH 0. == « "PCLAG: No input arg" DOERR » IFT         |                                |
| DUP TYPE 3. ≠ « "PCLAG: Invalid input type" DOERR » IFT |                                |
| <i>Save system flags and set CAS flags</i>              |                                |
| PUSH -3 CF -17 SF -103 SF -105 CF -120 SF               |                                |
| <i>Clear previous errors</i>                            |                                |
| ERR0                                                    |                                |
| <i>Get polynomial, with error trapping</i>              |                                |
| IFERR LAGRANGE THEN POP "PCLAG: " ERRM + DOERR          |                                |
| <i>No error; get polynomial coefficients</i>            |                                |
| ELSE POLYc                                              |                                |
| END                                                     |                                |
| <i>Restore system flags</i>                             |                                |
| POP                                                     |                                |
| »                                                       |                                |

## PCLEG Code – Coefficients of Legendre polynomial

PCLEG returns the coefficient vector for a Legendre polynomial of order  $n$ . The native LEGENDRE command is used, which runs without errors in Numeric Approx mode so the system flags need not be saved and restored.

---

|       |                              |
|-------|------------------------------|
| PCLEG | (177 bytes, checksum #671Ah) |
|-------|------------------------------|

---

```
«  
  Verify stack argument and validate argument type  
  DEPTH 0. == « "PCLEG: No input arg" DOERR » IFT  
  DUP TYPE 0. ≠ « "PCLEG: Invalid input type" DOERR » IFT  
  Clear previous errors  
  ERR0  
  Get polynomial, with error trapping  
  IFERR LEGENDRE  
  Handle error  
  THEN "PCLEG: " ERRM + DOERR  
  No error; find polynomial coefficients  
  ELSE POLYc  
  END  
»
```

---

## PCRAT Code – Coefficients of Taylor polynomial of rational polynomial

PCRAT calls the native DIVPC function to return the coefficient vector for the Taylor polynomial of a rational polynomial  $N(x)/D(x)$ . CAS system flags must be set and restored since DIVPC fails with CAS prompts in Numeric Approx mode.

---

|       |                              |
|-------|------------------------------|
| PCRAT | (286 bytes, checksum #B1F8h) |
|-------|------------------------------|

---

```
«  
  Verify stack arguments and validate arguments types  
  DEPTH 3. < « "PCRAT: <3 input args" DOERR » IFT  
  3 DUPN TYPE 0. ≠ ROT TYPE 9. ≠ ROT TYPE 9. ≠ OR OR  
  « "PCRAT: Invalid input arg(s)" DOERR » IFT  
  Clear previous errors  
  ERR0  
  Save system flags and set CAS flags  
  PUSH -3 CF -17 SF -103 SF -105 CF -120 SF  
  Get polynomial, with error trapping  
  IFERR DIVPC THEN POP "PCRAT: " ERRM + DOERR  
  No error; find polynomial coefficients. Restore system flags.  
  ELSE POLYc POP  
  END  
»
```

---

## PCTAY Code – Coefficients of Taylor polynomial of a polynomial

PCTAY calls the native PTAYL function to return the coefficient vector for the Taylor polynomial of a polynomial. CAS system flags must be set and restored since PTAYL raises CAS prompts in Numeric Approx mode.

---

|       |                              |
|-------|------------------------------|
| PCTAY | (266 bytes, checksum #2842h) |
|-------|------------------------------|

---

«

*Verify stack arguments and validate arguments types*

DEPTH 2. < « "PCTAY: <2 input args" DOERR » IFT

DUP2 TYPE 0. ≠ SWAP TYPE 9. ≠ OR

« "PCTAY: Invalid input arg(s)" DOERR » IFT

*Save system flags and set CAS flags*

PUSH -3 CF -17 SF -103 SF -105 CF -120 SF

*Clear previous errors*

ERR0

*Get polynomial, with error trapping*

IFERR PTAYL THEN POP "PCTAY: " ERRM + DOERR

*No error; find polynomial coefficients.*

ELSE POLYc

END

*Restore system flags*

POP

»

---