sas innovate 2026

Vectors and Complex Numbers in Base SAS

Building Functions for Vector and Complex Number Math using PROC FCMP

Paul McDonald

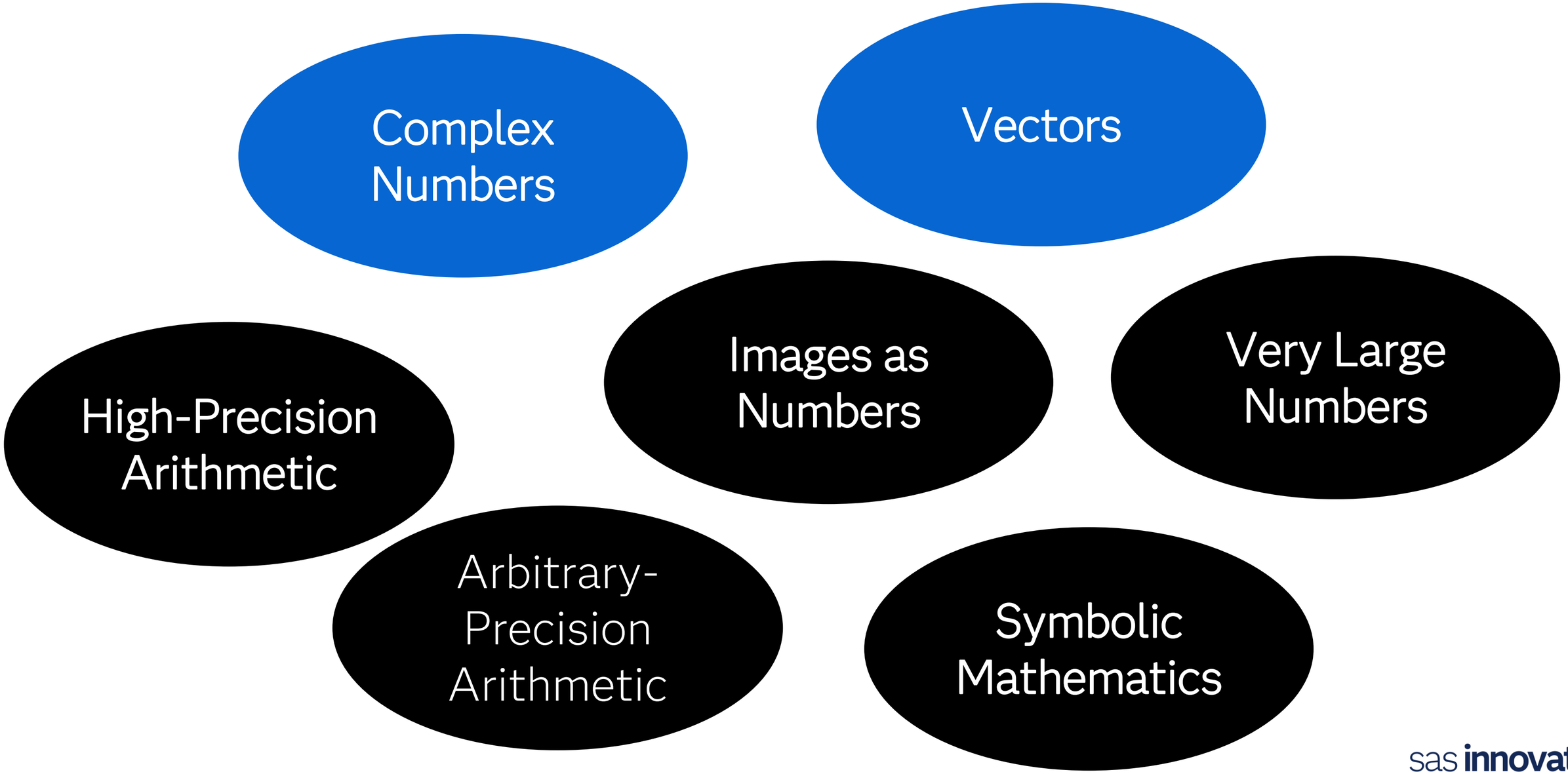
Start here...

Source Code for demos and ideas

The Problem

SAS handles data extremely well
SAS provides extensive support for math and statistics
But lacks in some areas…

Common Gaps



QUIZ

What do you do if SAS doesn't have something you need?

ANSWER

Build it yourself

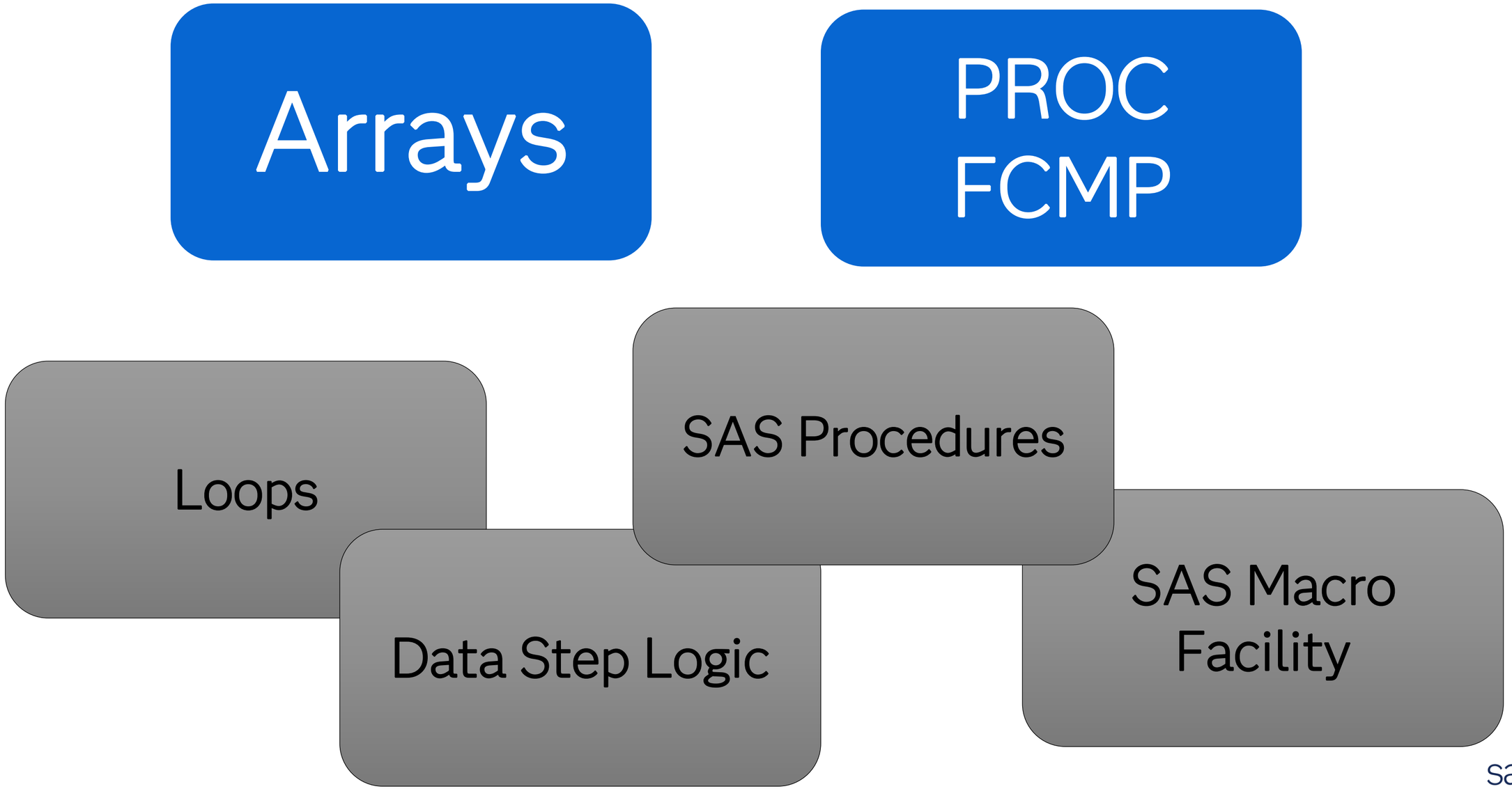
CALL SQRTNEG

CALL VECTORSUM

VECTORMAG()

DOTPROD()

What SAS does give us



Why Arrays?

Maintain an ordered list of values

Why PROC FCMP?

Create custom functions and call routines

Why not SAS Macros?

Processing takes place in the data step for each record

$a + bi$

$z = r(\cos\varphi + i\sin\varphi)$

(a, b)

Complex Numbers

Let’s take the square root of minus one

$z = r\angle\varphi$

$\begin{pmatrix} a & -b \\ b & a \end{pmatrix}$

$$\sqrt{-1} = i$$

Unless you are an Electrical Engineer… then:

$$\sqrt{-1} = j$$

Run In Base SAS

```
data _null_ ;
    R = -1 ;
    C = sqrt(R) ;
    put R= C= ;
run ;
```

Result in SAS Log

```
69 data _null_ ;
70 R = -1 ;
71 C = sqrt(R) ;
72 put R= C= ;
73 run ;
NOTE: Invalid argument to function SQRT(-1) at line 71 column 8.
R=-1 C=.
R=-1 C=. _ERROR_=1 _N_=1
NOTE: Mathematical operations could not be performed at the following
      places. The results of the operations have been set to missing
      values.
Each place is given by: (Number of times) at (Line):(Column).
1 at 71:8
```

Because…

Syntax

SQRT(*argument*)

Required Argument

argument

specifies a numeric constant, variable, or expression. *Argument* must be nonnegative.

Complex Numbers are made of two components

$$(Complex) = (Real) + (Imaginary)x\ i$$

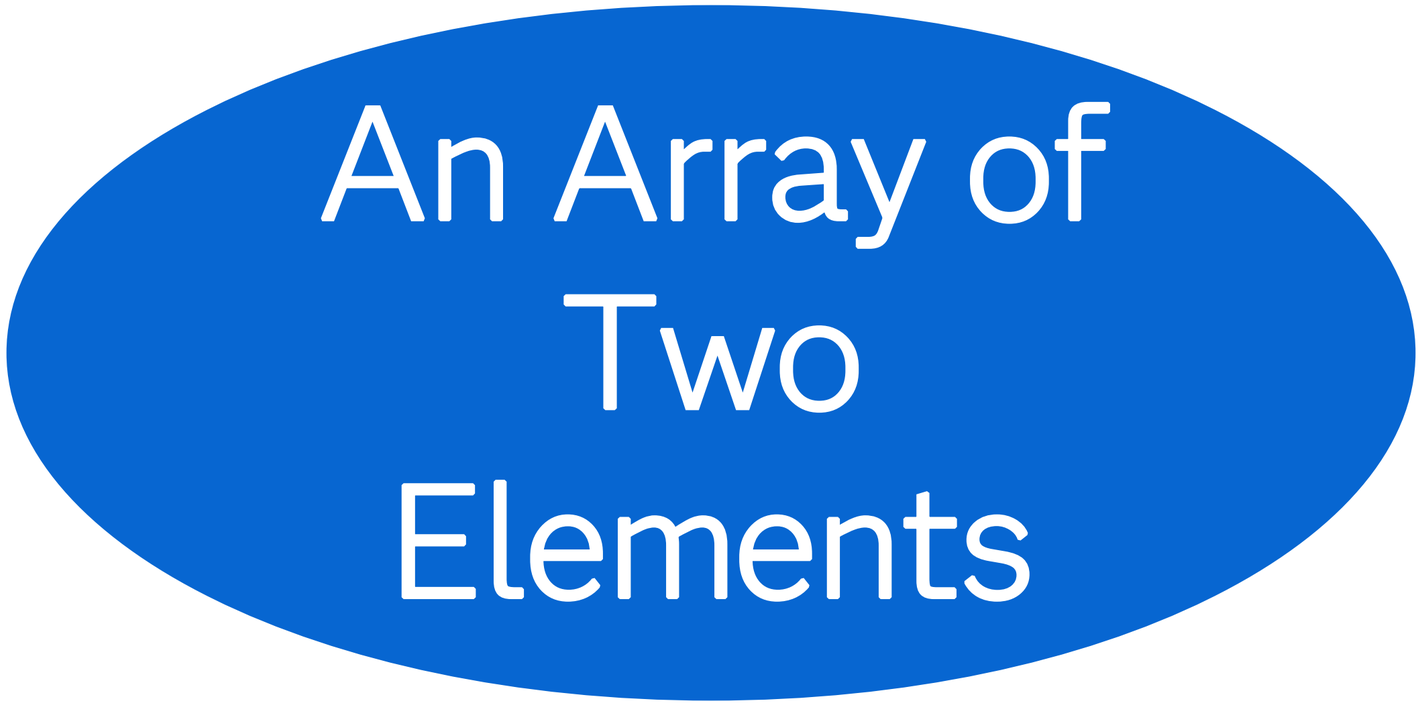
Often expressed as

$$C = a + bi$$

Cartesian expression

Polar and other styles… later!

Complex Numbers are essentially…



CALL SQRTNEG

Assigns the square root of a number as a two-dimensional array in the form a + b*i*.

CALL SQRTNEG (*argument*, *array-name*) ;

Required Arguments

argument

specifies a numeric constant, variable, or expression.

array-name

specifies the name of an array that was previously defined in the same DATA step. This argument cannot be a constant, variable, or expression. The array must be a two-dimensional numeric array for a cartesian complex number in the form *array-name* = *array-name*[1] + *array-name*[2] * *i*.

Make CALL SQRTNEG

```
proc fcmp outlib=work.funcs.math ;
  subroutine sqrtneg (real, complex[*]) ;
      outargs complex ;
      if real = . then do ;
         complex [1] = . ;
         complex [2] = . ;
      end ;
      else if real < 0 then do ;
         complex [1] = 0 ;
         complex [2] = sqrt(abs(real)) ;
      end ;
      else do ;
         complex [1] = sqrt(real) ;
         complex [2] = 0 ;
      end ;
   endsub ;
run ;
```

Let’s Do It

$$C = \sqrt{x} = a + bi$$

```
data _null_ ;
    set complexdemo ;
    array  C (2) a b ;
    call sqrtneg (x, C) ;
    put x= a= b= ;
run ;
```

```
69          data _null_ ;
70              set complexdemo ;
71              array  C (2) a b ;
72              call sqrtneg (x, C) ;
73              put x= a= b= ;
74          run ;

x=-2 a=0 b=1.4142135624
x=-1 a=0 b=1
x=0 a=0 b=0
x=1 a=1 b=0
x=2 a=1.4142135624 b=0
```

Right Answers… but it looks kind of silly

```
x=-2 a=0 b=1.4142135624
x=-1 a=0 b=1
x=0 a=0 b=0
x=1 a=1 b=0
x=2 a=1.4142135624 b=0
```

COMPLEX2CHAR function

Returns a cartesian complex number array to a human-readable character string expressed as *a + bi*.

COMPLEX2CHAR (*array-name*) ;

Required Argument

array-name

specifies the name of an array that was previously defined in the same DATA step. This argument cannot be a constant, variable, or expression. The array must be a two-dimensional numeric array for a cartesian complex number in the form *array-name* = *array-name*[1] + *array-name*[2] * *i*.

Make COMPLEX2CHAR

```
proc fcmp outlib=work.funcs.math ;
   function complex2char(complex[2]) $32 ;
      return(strip(put(complex[1], best.))||
                ' + '||
                strip(put(complex[2], best.))||
                'i') ;
   endsub ;
run ;
```

```
data _null_ ;
    set complexdemo ;
    array  C (2) a b ;
    call sqrtneg (x, C) ;
    complex = complex2char(C) ;
    put complex= ;
run ;
```

Run this

Get that

```
complex=0 + 1.4142135624i
complex=0 + 1i
complex=0 + 0i
complex=1 + 0i
complex=1.4142135624 + 0i
```

Congratulations!

You just made a function and a CALL routine

CALL SQRTNEG

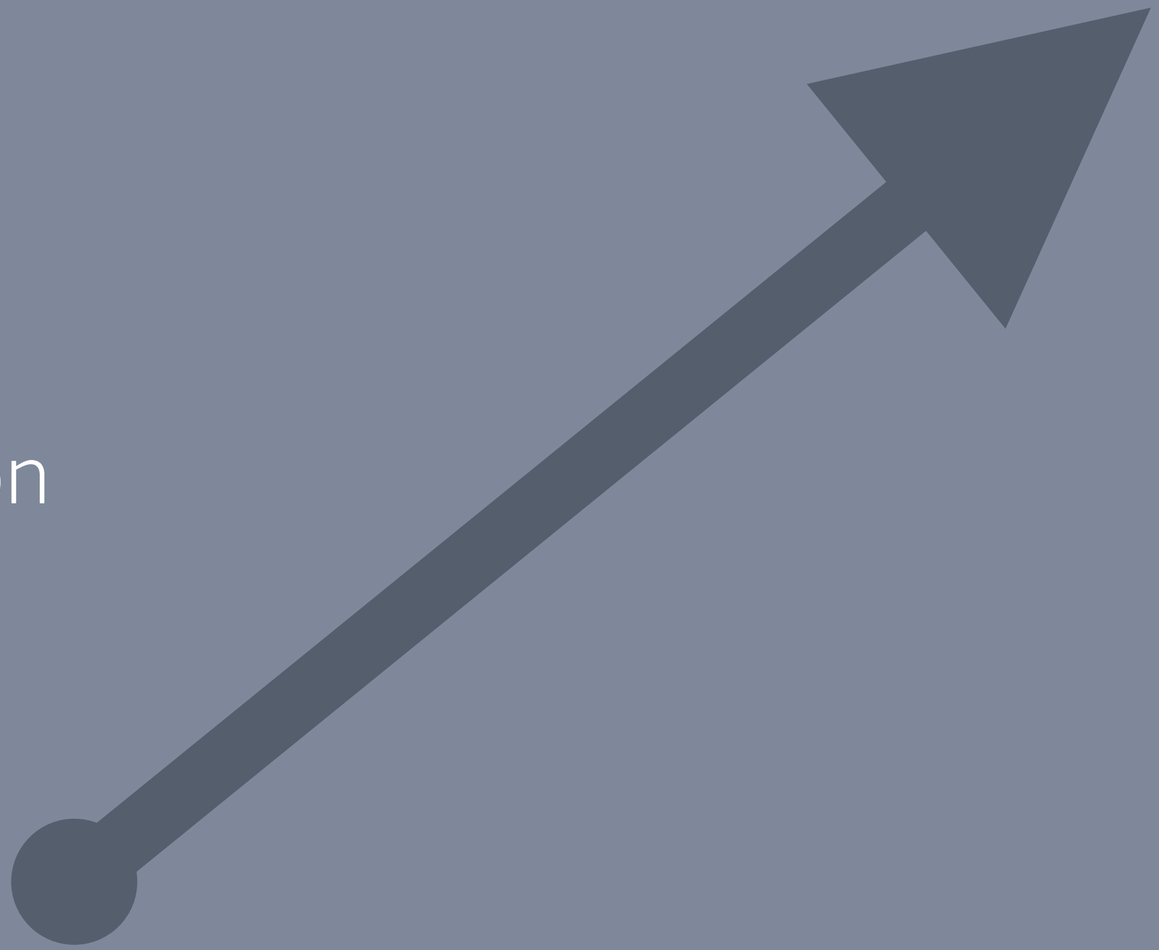
- Performs the square root and returns a complex number
- INPUT: Scalar value as a SAS Number
- OUTPUT: Complex number expressed as a SAS Array

COMPLEX2CHAR

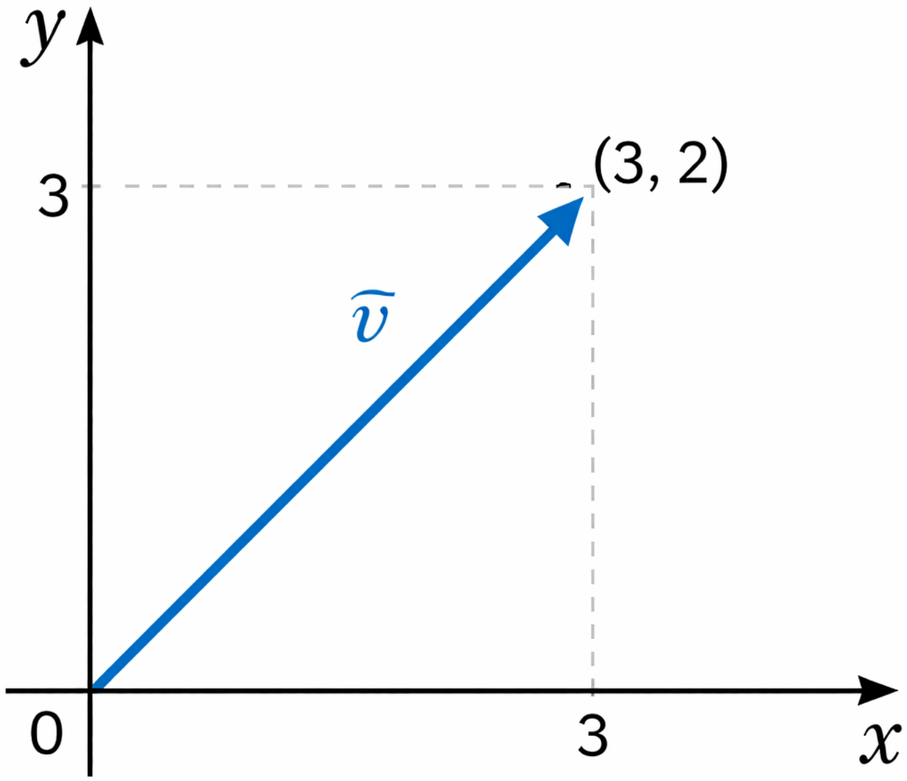
- Returns a cartesian complex number array to a human-readable character string expressed as a + bi
- INPUT: Complex number as an Array
- OUTPUT: Character string of complex number in human-readable format

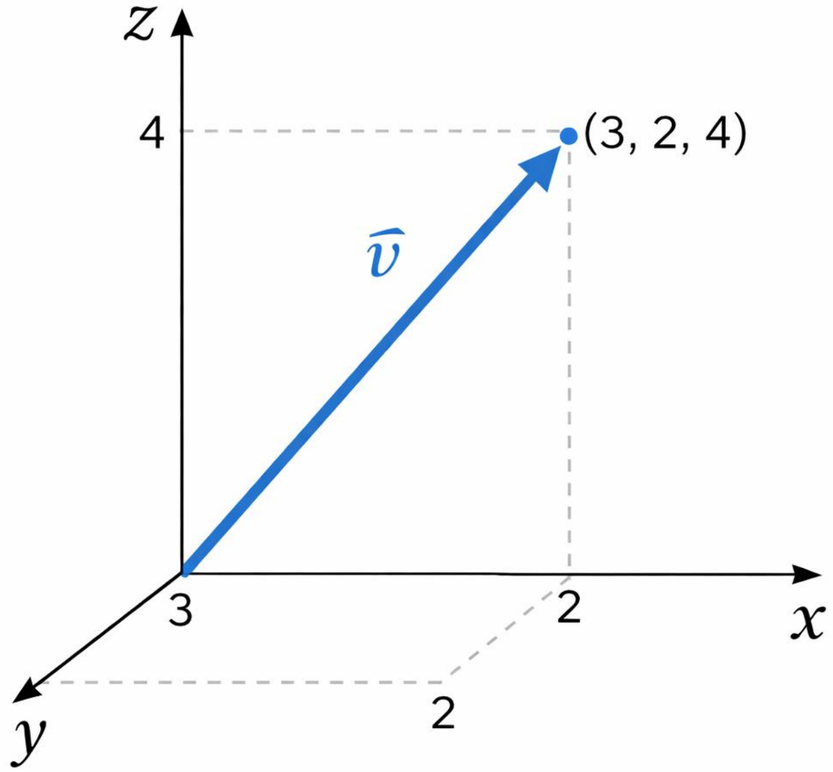
Vectors

Magnitude and direction



Vectors (2D and 3D)



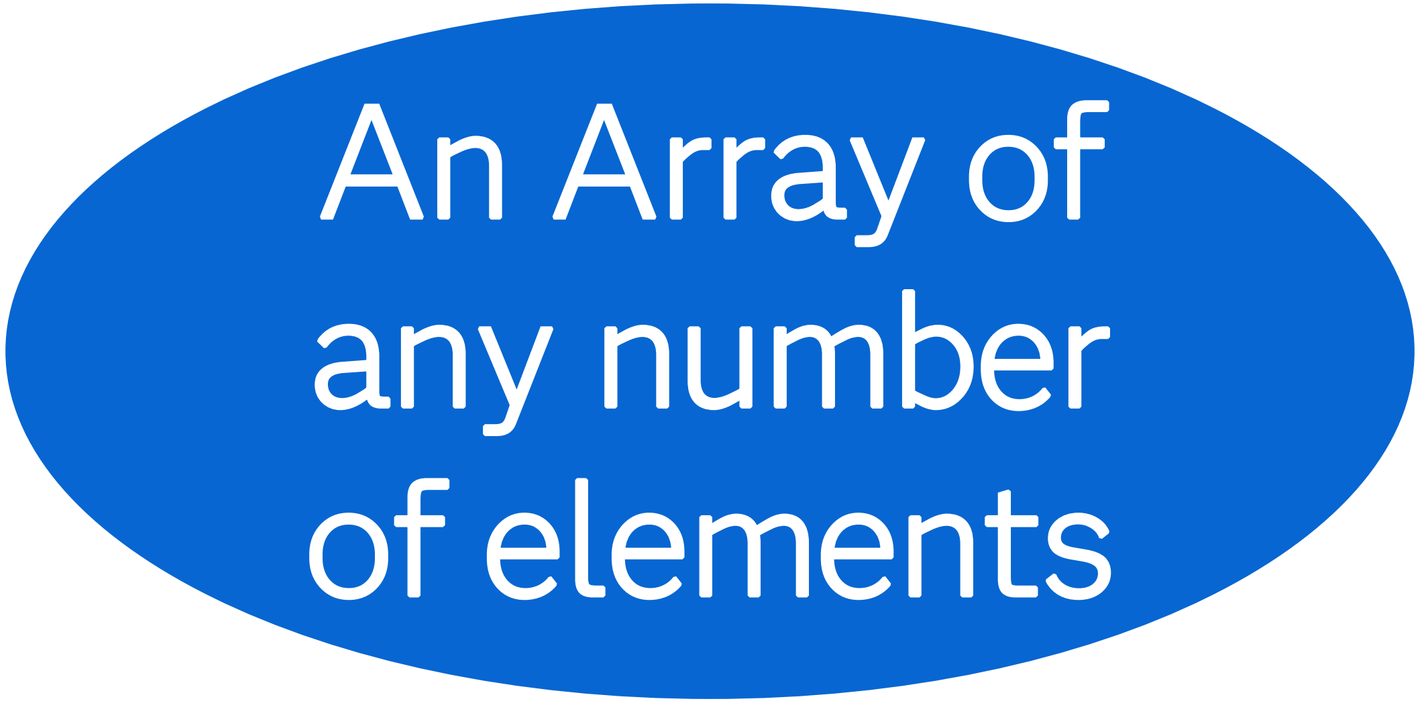


And even more dimensions if you want!

Vector Representations

Cartesian	(x, y, z)
Polar	(r, θ)
Cylindrical	(r, θ, z)
Spherical	(r, θ, ϕ)
Component	<x, y, z>
Unit Vector	xi + yj + zk
Matrix	$[x\ y\ z]^T$

Vectors are essentially…



VECTORMAG

- Calculates the magnitude of a vector
- INPUT: Vector as a SAS Array
- OUTPUT: Scalar value as a SAS Number

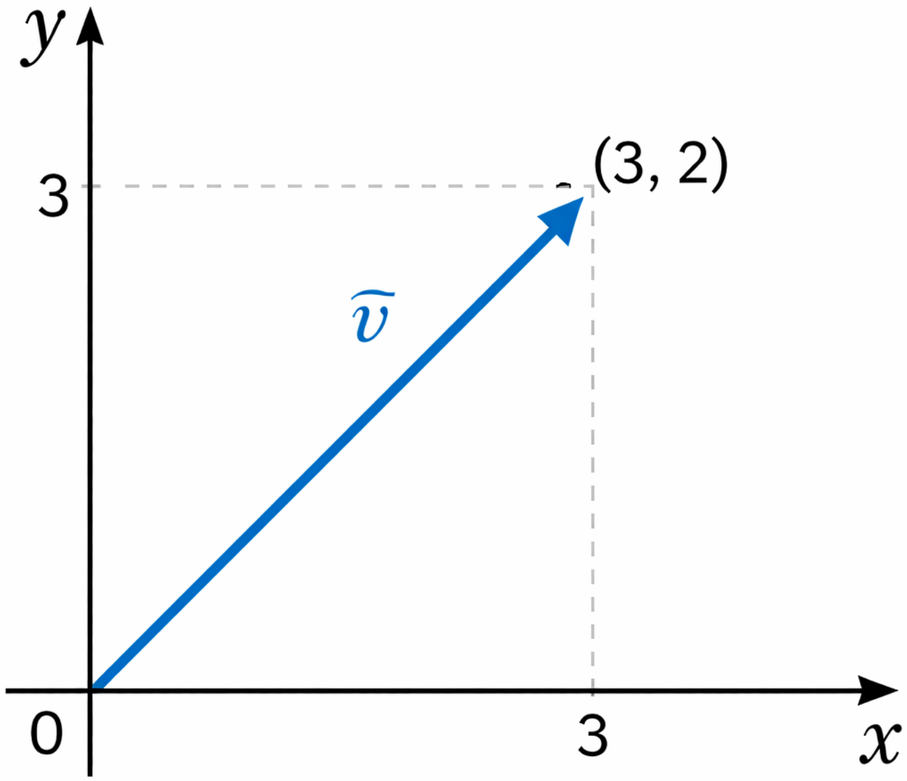
DOTPROD

- Calculates the Dot Product of two vectors
- INPUT: Vector as a SAS Array
- OUTPUT: Scalar value as a SAS Number

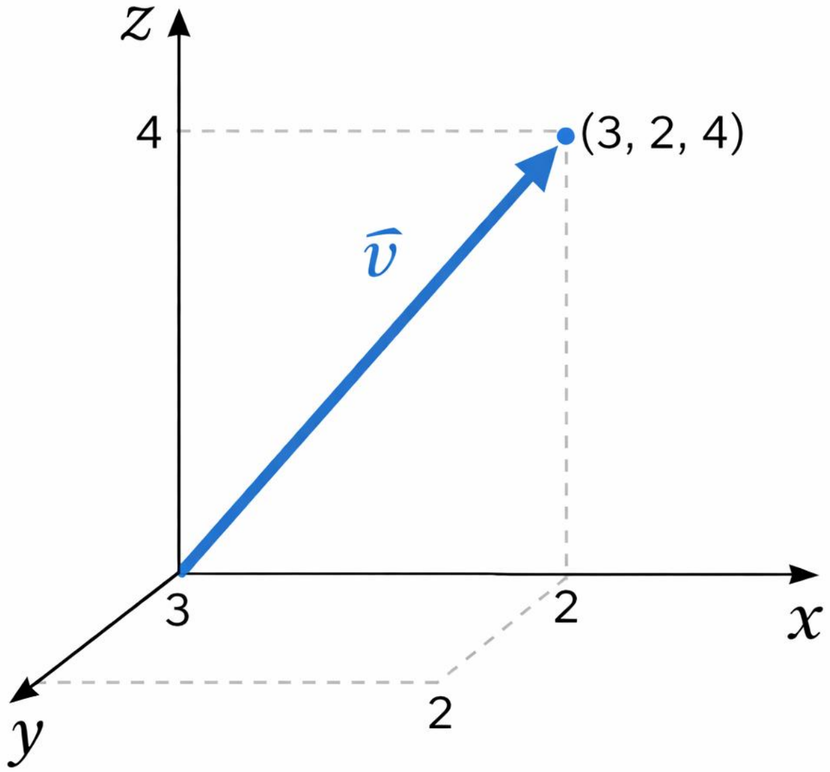
CALL VECTORSUM

- INPUT: three vectors as SAS Arrays
- Calculates the SUM of the first two vectors and writes the value to the third vector

Use Cartesian Representation



$\vec{V} = (3, 2)$



$\vec{V} = (3, 2, 4)$

```
data _null_ ;
    set vectordemo ;

    array a (5) a1-a5 ;
    array b (5) b1-b5 ;
    array c (5) c1-c5 ;

    magnitude_a = vectormag (a) ;
    call vectorsum (a, b, c) ;
    dot_product_a_b = dotprod (a, b) ;

    put magnitude_a= ;
    put c1= c2= c3= c4= c5= ;
    put dot_product_a_b= ;
run ;
```

```
proc fcmp outlib=work.funcs.math ;
   function vectormag (vector[*]) ;
      radicand = 0 ;
      do i = 1 to dim(vector) ;
         radicand + vector[i]**2 ;
      end ;
      return (sqrt(radicand)) ;
   endsub ;
run ;
```

```
proc fcmp outlib=work.funcs.math ;
   subroutine vectorsum(augend[*], addend[*], sum[*]) ;
      outargs sum ;

      if dim(augend)=dim(addend) and dim(addend)=dim(sum) then do i = 1 to dim(sum) ;
         sum[i] = augend[i] + addend[i] ;
      end ;
      else do i = 1 to dim(sum) ;
         sum[i] = . ;
      end ;
   endsub ;
run ;
```

```
proc fcmp outlib=work.funcs.math ;
   function dotprod(vector1[*], vector2[*]) ;
      if dim(vector1) ne dim(vector2) then return (.) ;
      total = 0 ;
      do i = 1 to dim(vector1) ;
         total + vector1[i] * vector2[i] ;
      end ;

      return (total) ;
   endsub ;
run ;
```

```
data _null_ ;
   set vectordemo ;

   array a (5) a1-a5 ;
   array b (5) b1-b5 ;
   array c (5) c1-c5 ;

   magnitude_a = vectormag (a) ;
   call vectorsum (a, b, c) ;
   dot_product_a_b = dotprod (a, b) ;

   put magnitude_a= ;
   put c1= c2= c3= c4= c5= ;
   put dot_product_a_b= ;
run ;
```

Run this

Get that

```
magnitude_a=12.409673646
c1=11 c2=8 c3=13 c4=12 c5=16
dot_product_a_b=156
```

Congratulations!

You just made two more functions and another CALL routine

What’s next?

Where will this take you?

Add QC Checks

Edge cases

More Functions and Routines

Defaults and overrides

Volume Testing

WARNING: ERROR:

Usage Guides and Support

Practical Applications

How can you use this?

Signal and frequency analysis

Distance models

Multi-variable probability

Movement and direction forecasts

Joint distributions

3D positioning

Wave systems

Your turn

Get ready to share ideas!

SAS Provides the Building Blocks

You build what you need

What are some functions and routines that you wish SAS would add?

60 second brainstorm

Brainstorm Share

I wish I could do _____ in SAS

You can do this all yourself…

…Because you are smarter than you think you are!

In case you missed it...

Source Code for demos and ideas

Vectors and Complex Numbers in Base SAS

paulmcdonald@sbcglobal.net

sas innovate 2026