Solvers

Solving Equations by programming.

Author: Dara O Shayda

Artwork: The Electrofishers were originally Planters who have splintered off and forgotten their past. They currently reside in an Abandoned Level after their original home was destroyed in a Safeguard Attack.

- Preface
- Background
  - Terminologies
  - Symbols
  - Constants
  - Operators
  - ==
  - Expressions
  - Words
  - Phrases
  - Sentences
- Sides of Equations
  - Manual Operations on the Sides
  - Right hand side vs. Left hand side
  - Subtract from Sides
  - Add to Sides
  - Divide both Sides
  - Multiply both Sides
- solve[ ]
  - solve a*x + b = 0
  - solve a*x +b == 0 for x

- instance [ ]

Preface

This mostly **live-code** computational Wiki book addresses two audiences:

1. STEM high school students to undergraduate college level students
2. Programmers' and computer scientists' **learning habitat** to freely experiment with new concepts in programming languages

On one hand the book is regimented for the absolute newbies with no programming experience, and on the other hand pushes the programmers off the cliff of commercial programming languages for better experiences.

I am borrowing components from the formal language of the Alfred Tarski on modern mathematical logic. I did that to make my language more consistent with a concise paradigm + a terrific economy of the words, while staying away from the fluffy contemporary foo-foo dust publications.

Dara O Shayda

Chief of Software

Computational Classnotes

Republic of Ireland, May 8th 2026

foo-foo dust : A reference to a (non-existent) power or mysterious ingredient or hidden effort that creates desirable results. Results achieved, as if by magic, perhaps by slight-of-hand.

https://www.urbandictionary.com/define.php?term=foo+foo+dust

Background

Basics for setting up equations and extracting solutions.

Terminologies

lhs : Left Hand Side

rhs : Right Hand Side

p = q : an identity

Equational Expression: expr1 = expr2 e.g. expr1 = a*x+b and expr2 = cos[b-c]

copy: when an object is duplicated, byte by byte so to say, into a facsimile.

> This definition of **copy** is the simplest to grasp the concept. In due places, more sophisticated definitions will be produced.

Structure: a **structure** is that which permitting **Similarity** to other **Similars**.

Example: **Circular** is a structure since it permits **similarity to** anything that is **round**.

Example: **Red** is a structure since it permits **similarity to** anything that is **red**.

Example: **Numeral** is a structure since it permits **similarity to** anything that is a **number**.

ff calls a list e.g. **{1,2,3}** as a **structure**, or a function e.g. **cos[ ]** a **structure** or an **algebraic structure** e.g. **x^2-1**.

Background

Symbols

In Free Form Programming Language (ff) the concept of Symbol is more than one:

1. Individual alphabet in a language e.g. x in English , ω in Greek
2. Composite of individual symbols e.g. dara that while it is the concatenation of four alphabets ff treats it as a single symbol
3. Mathematical symbols e.g. ⊗
4. Glyph (Unicode) e.g. □□ , □□
5. Glyph a raster image (this option is currently unavailable for HTML use)

> The mathematical symbols in many cases have particular syntax and semantics in an expression which cannot be altered or ignored.

> The mathematical symbol single space or " " is interpreted multiplication in mathematical textbooks. Additionally in ff often " " is assigned the functionality of separator in a list of items.

o be more general ff defines a Symbol as follows:

> An atomic or indivisible structure with no discerning programmable components is called a Symbol.

Background

Constants

In Free Form Programming Language (ff) the concept of a **Constant** is an **immutable** structure.

Examples : 5, □□ .

Background

Operators

=

+

-

*

/

{}

()

;

""

->

<

>

<=

∩

∪

⊕

×

∂

∈

∉

⊂

→

and

Background

〓

value to rhs. What do we mean equal by value? Example: 3 = 6/2 in spite of the fact that lhs and rhs are made from very different expressions but their values are Identical or the same, hence the Identity term.

= : copies stores the rhs into lhs;

```
// = stores the rhs into lhs;
x1  = 5;

x2 = 2 * x1;

show x2;

// == creates an equations between lsh and rhs;
x3 = (x1 == 5);

show x3;



x4 = (x2 == 10);

show x4;

// lhs and rhs with different variables;
x5 = (x2 == 2*x1);

show x5;

save as equations;
```

Background

Expressions

Algebraic

a*(c-u)

Free Form

solve a*x +b == 0 for x

Background

Words

"I am a word"

Background

Phrases

The highlighted are the phrases

pts = evaluate l2 in the -1 to 3 range at increments of (1/4);

Part of a sentence, have a meaning in and of itself, but often has no procedural meaning alone.

Background

Sentences

pts = evaluate l2 in the -1 to 3 range at increments of (1/4);

> FIXME: line cannot be even be used as a subphrase;

l2 := l[v,t,{0,b}];

Sides of Equations

Applying same operations on both sides of =

Manual Operations on the Sides

> **What**: Use ff to duplicate the manual ways handling equations as taught in school teachings.
> **Why**: To make sure you understand how to manipulate sides of an equation on your own.
> **How**: Simply try to duplicate what the teacher does to solve an equation by working the sides.
> **Time To Complete**: 3 hours.

Manully set the lhs and rhs variables to some expressions of choice and then commence, step by step, applying the same arithmetic operations to both sides.

In this example, the equation

x-3 = 17***y*** +9

is being solved for **y** namely the equation is transformed to a new equation with **y** alone on either side as the solution.

```
lhs = x - 3;
rhs = 17 * y + 9;

lhs = lhs - 9;
rhs = rhs - 9;

lhs = lhs / 17;
rhs = rhs / 17;

show lhs also rhs;

save as sides;
```

Output

"lhs" → (-12 + x)/17
"rhs" → y

Right hand side vs. Left hand side

> **What**: Different sides to an operator e.g. ==
> **Why**: Proper treatment of the sides of an operator e.g. == results in much functionality
> **Time To Complete**: 3 hours

.left and .right

In Free Form Programming Language (ff) the dot operator, namely the dot on your keyboard, allows for accessing **parts** of an expression!

For a given equation e.g. eq, in the ff program below, the lhs or x^2-1+y can be accessed by the expression eq.left and a/b+c by the expression eq.right.

```
eq = (x^2-1+y == a/b+c);

lhs = eq.left;

show lhs;

rhs = eq.right;

show rhs;

save as sides;
```

Output

```
"lhs" → -1 + x^2 + y
"rhs" → a/b + c
```

The programmer is not required to use a symbol e.g. eq for this . access, as long as using the ( ) operator the results would be the same:

```
tmp = (x^2-1+y == a/b+c).left;

show tmp;

save as lhsrhs;
```

Output

"tmp" → -1 + x^2 + y

?

In Free Form Programming Language (ff), the Symbol ⊕ is one of several unassigned operators.

> As you can see the . operator works perfectly even if the operator in use is not defined! This is a core language aspect of ff that allows undefined variables as well as undefined functions and undefined operators alike.

```
eq = (x^2-1+y)  ⊕  (a/b+c);

lhs = eq.left;

show lhs;

rhs = eq.right;

show rhs;

save as sides;
```

Output

"lhs" → -1 + x^2 + y
"rhs" → a/b + c

Try different operators.

```
eq = (x^2-1+y)  <=  (a/b+c);

lhs = eq.left;

show lhs;

rhs = eq.right;

show rhs;

save as sides;
```

Output

"lhs" → -1 + x^2 + y
"rhs" → a/b + c

Subtract from Sides

> **What**: Subtract the same expression from the both sides of ==
> **Why**: Such side-wise subtraction is an essential computation for solving equations
> **Time to Complete**: 3 hours

```
eq = (x -3 == 17*y + 9);

lhs = eq.left - 9;
rhs = eq.right - 9;

eq2 = (lhs == rhs);

show eq2;

save as sides;
```

Output

"eq2" → -12 + x == 17*y

Output

"lhs" → (-12 + x)/17
"rhs" → y

Add to Sides

> **What**: Add the same expression to the both sides of == .
> **Why**: Such side-wise addition is an essential computation for solving equations.
> **Time to Complete**: 3 hours.

```
eq = (x -3 == 17*y - 9);

lhs = eq.left + 9;
rhs = eq.right + 9;

eq2 = (lhs == rhs);

show eq2;
```

Output

"eq2" → 6 + x == 17y

Divide both Sides

> **What**: Divide both sides of == by the same expression.
> **Why**: Such side-wise division is an essential computation for solving equations.
> **Time to Complete**: 3 hours.

```
eq = (x - 3 == 17*y);


lhs = eq.left / 17;
rhs = eq.right /17;


eq2 = (lhs == rhs);


show eq2;


save as lhsrhs;
```

Output

"eq2" → (-3 + x)/17 == y

Multiply both Sides

> **What**: Multiply both sides of == by the same expression.
> **Why**: Such side-wise multiplication is an essential computation for solving equations.
> **Time to Complete**: 4 hours.

```
eq = (x - 3 == y / 13);

lhs = eq.left * 13;
rhs = eq.right * 13;

eq2 = (lhs == rhs);

show eq2;
```

Output

"eq2" → 13*(-3 + x) == y

Output

"eq2" → 13*(-3 + x) == y

Apply expand[ ] to one side

```
eq = (x - 3 == y / 13);

lhs = eq.left * 13;
rhs = eq.right * 13;

eq2 = (lhs == rhs);

show eq2;
```

```
lhs = expand[eq2.left];
rhs = eq2.right;

eq3 = (lhs == rhs);

show eq3;

save as lhsrhs;
```

Output

"eq2" → 13*(-3 + x) == y
"eq3" → -39 + 13*x == y

---

solve[ ]

solve [ ] function

solve[ ]

solve a*x + b = 0

```
sol =  solve[ a*x + b == 0, {x}];


show sol;


save as solve4;
```

Output

"sol" → {-(b/a)}

solve[ ]

solve a*x +b == 0 for x

```
sol =  solve a*x +b == 0 for x;


show sol;


save as solve4;
```

Output

"sol" → {-(b/a)}

solve[ ]

instance [ ]

```
radius=norm[{x,y}];
linear = 3*x+2*y;

pts = instance [linear <= 3 and 0.5<=radius <=1 , 300];

//show pts;
pointplot pts;

save as cropped_anulus;
```

Ouput