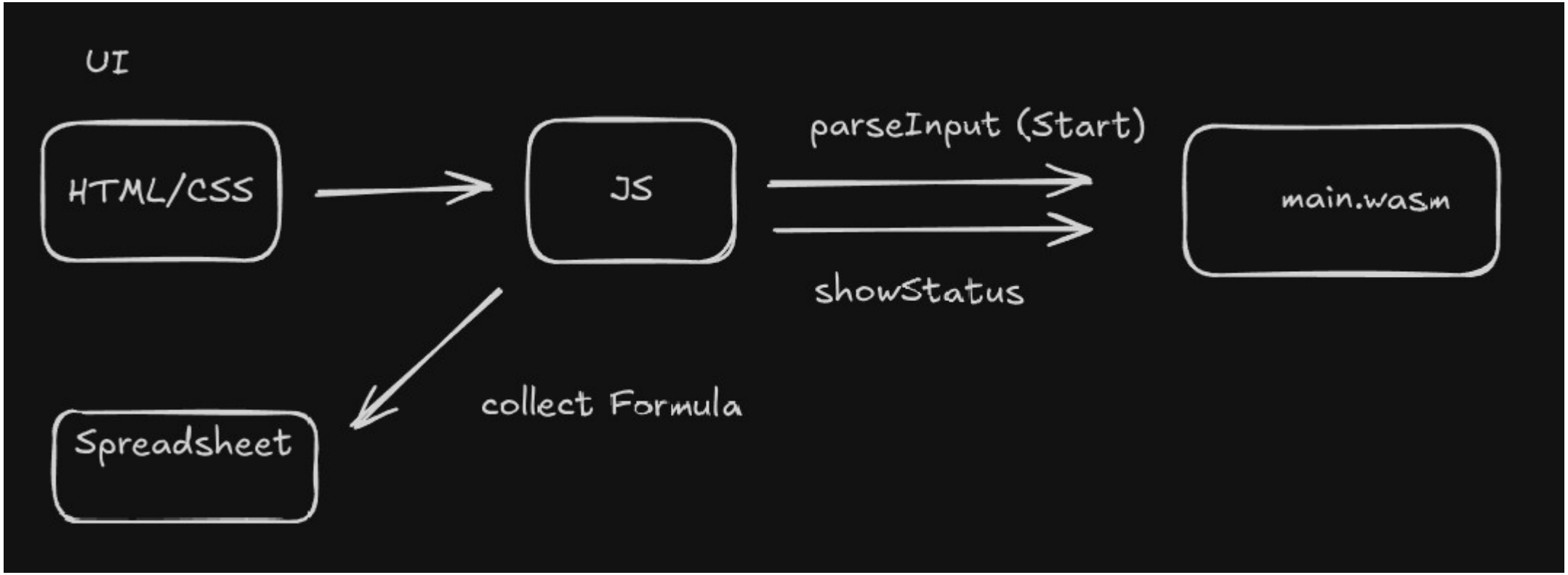
Only Office NLP Solver

Content

- Introducing OnlyOffice and showing the Solver
- Overview architecture, flow of data
- Getting formulas and user input in JS
- Preparing Go for WASM, connect to JS
- Dynamic Equations with Tokenizer and Parser -> Eval func
- Finally the Solver. Custom Downhill Simplex
- State of the project and next steps

Overview architecture, flow of data



karlbreuer.com

Getting formulas and user input in JS

1.Collect Formula (Show in spreadsheet)

2. Parse Input

"((B28*0.08)+(B29*0.06)+(B30*0.42)+(B31*0.22))"

And the start params!

```
type StartParamsT struct {
        Cell     string  `json:"cell"`
        MinValue float64 `json:"min_value"`
        MaxValue float64 `json:"max_value"`
        LimitMin bool    `json:"limit_min"`
        LimitMax bool    `json:"limit_max"`
}
```

Preparing Go for WASM, connect to JS

```
//go:build js && wasm
// +build js,wasm

package main


func main() {
    // Keep the program running
    c := make(chan struct{})

    // Register the addOne function to be callable from JavaScript
    js.Global().Set("goAddOne", js.FuncOf(addOneJS))
    js.Global().Set("goGetResult", js.FuncOf(getResultJS))
    js.Global().Set("goParseInput", js.FuncOf(parseInputJS))
    js.Global().Set("goCheckStatus", js.FuncOf(checkStatusJS))

    // Block forever
    <-c
}
```

```
//go:build !wasm
// +build !wasm

package main
```

$GOOS=js GOARCH=wasm go build -o main.wasm .

Preparing Go for WASM, connect to JS

```
<script src="scripts/wasm_exec.js"></script>
<script src="scripts/go-script.js"></script>
```

```
//go-script.js
(function () {
  "use strict";

  let wasmReady = false;

  // Load Go WASM
  async function loadWasm() {
    try {
      const go = new Go();
      const result = await WebAssembly.instantiateStreaming(
        fetch("go/main.wasm"),
        go.importObject
      );

      // Run the Go program
      go.run(result.instance);

      wasmReady = true;

    } catch (err) {

      console.error("WASM load error:", err);
    }
  }

  // Initialize WASM when plugin is ready
  window.addEventListener("DOMContentLoaded", function () {
    // Wait a bit for Asc.plugin to be available
    setTimeout(loadWasm, 100);
  });

  // Also expose loadWasm for manual initialization if needed
  window.loadGoWasm = loadWasm;
})();
```

```
window.goParseInput(
            res.targetFormula,
            paramsJSON,
            iterations.toString(),
            sideConditionsJSON
          );
```

Dynamic Equations with Tokenizer and Parser -> Eval func

The Problem: We get a string, and have **no chance to know the equation at compile time.**

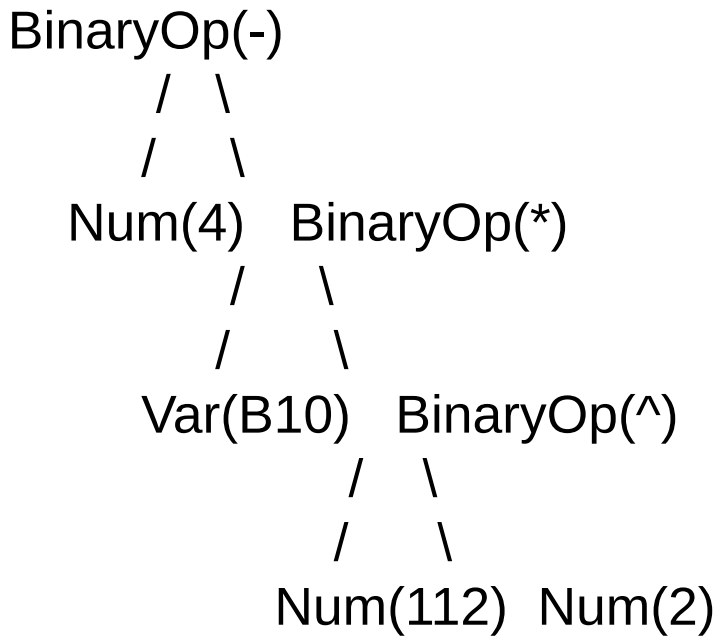
We have to build the evaluation function for the equation dynamically at runtime instead

Input = **"(4-B10*112^2)"**

First step tokenize!

"(4-B10*112^2)" → [(, 4, -, B10, *, 112, ^, 2,)]

Second step is building the node tree structure, higher index gets evaluated first!



The Solver

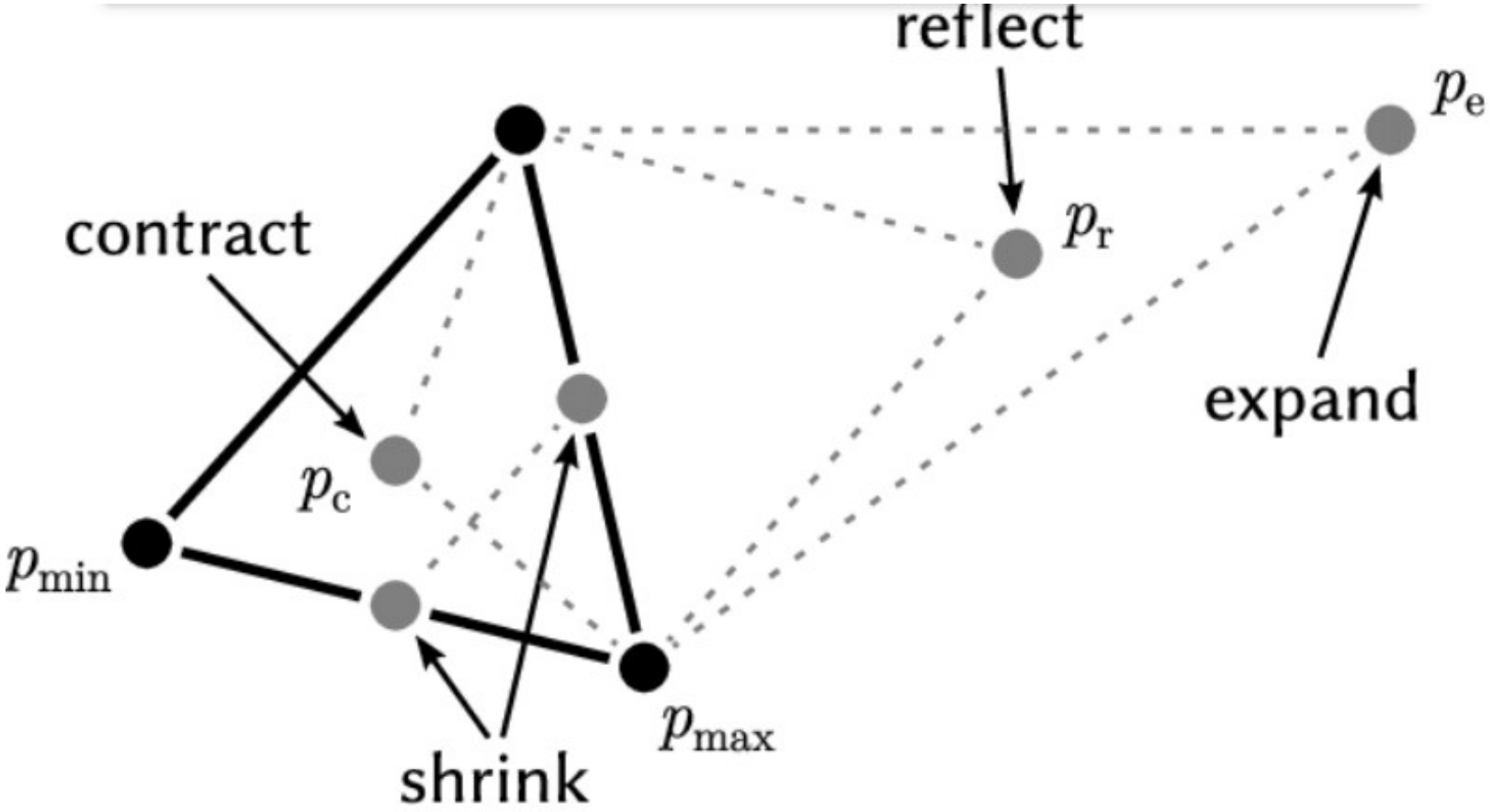
The solver itself is nothing more than trying out a lot of possible values for all parameters in a smart way!

I used a custom variation of the nelder mead downhill simplex, so that we can theoretically fit any function that we can calculate

There are 2 important things left:.

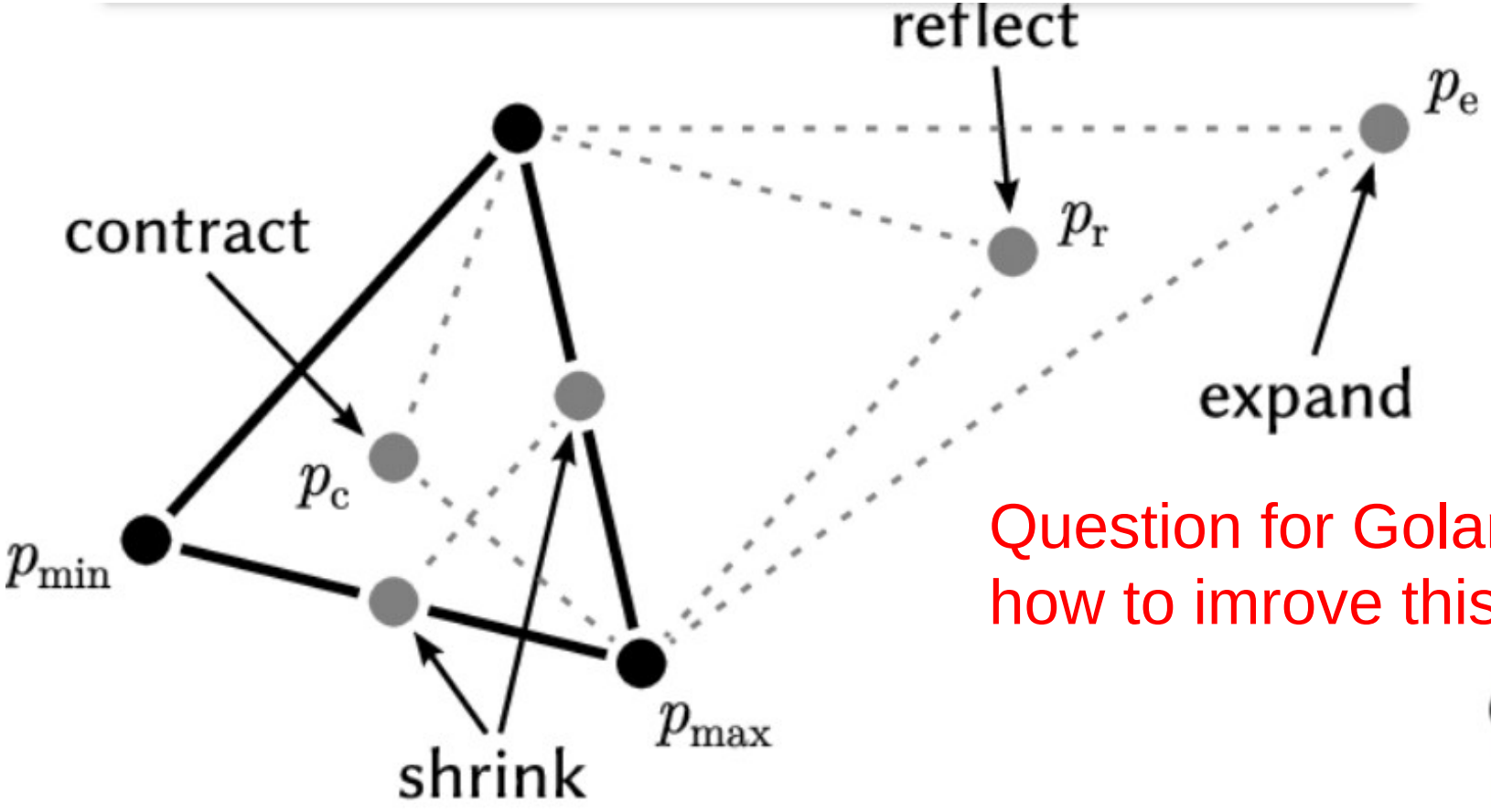
The algorithm itself, and a good **starting simplex**.

The Solver – The algorithm



https://en.wikipedia.org/wiki/Nelder%E2%80%93Mead_method

The Solver – The algorithm

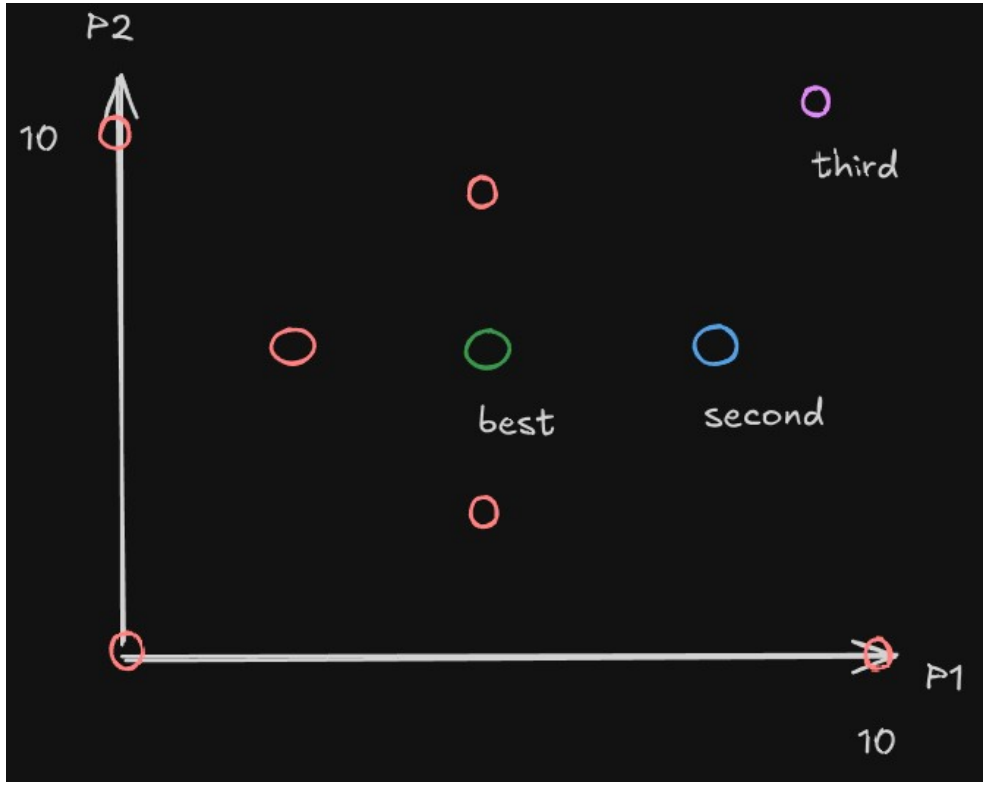


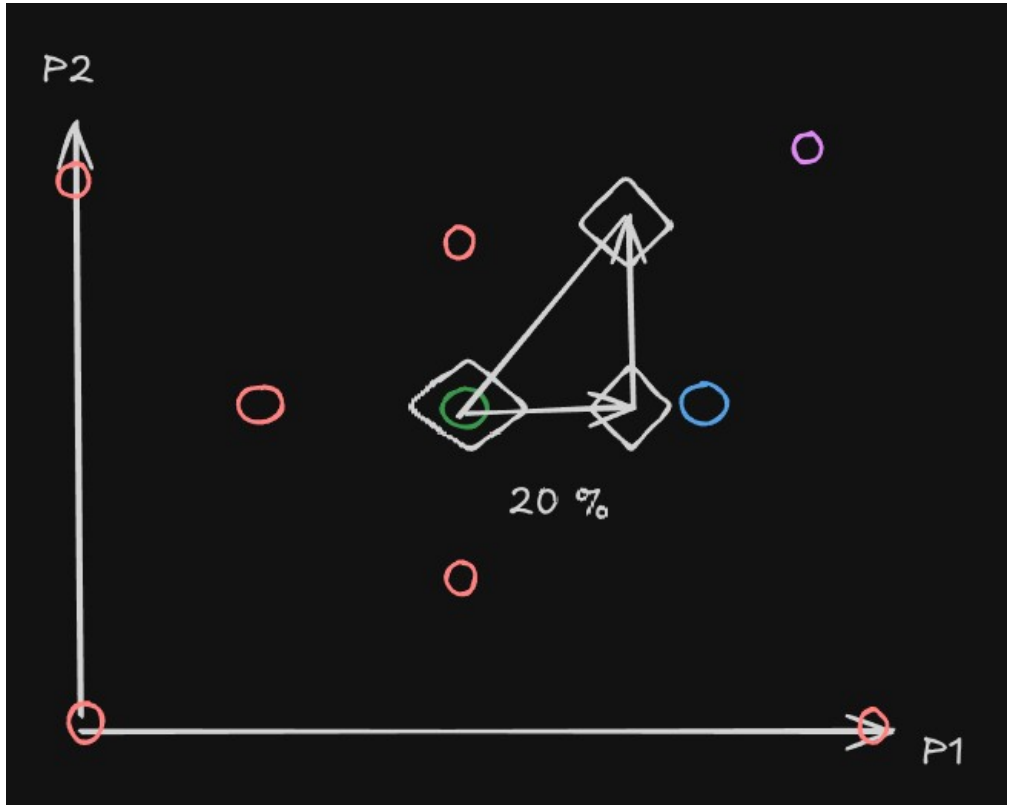
Question for Golang audience
how to imrove this?

https://en.wikipedia.org/wiki/Nelder%E2%80%93Mead_method

The Solver – The starting simplex

We use the starting value from the user to specify the range of the starting simplex





https://en.wikipedia.org/wiki/Nelder%E2%80%93Mead_method

Next steps

Add an addition **linear** solver, that works better with linear problems and side condition!

Listen for actual user input. Bugs and use-cases!