4. EQUATION SOLVING

c) Advanced system solver

November 27, 2021

Support functions...

[10]:
```
import numpy as np
# Reads the equation strings and converts them into lists of various values...
def getLists(equationList):
    var, coef, const, varCount, eqCount = {}, [], [], 0, 0
    # NOTES:
    # 'coef' is a list of lists.
    # Each sublist is for a variable, each sublist element corresponds to the␣
↪equation number.
    # 'var' is the dictionary, with key as variable name and value as the␣
↪variable index.
    # Variable indices are simply to help associate the coefficient lists to␣
↪the variables.
    for e in equationList:
        e, i, varsFound, sign = e + "\\", 0, [], 1
        # If coefficient of a variable is to the right of "=", we will take it␣
↪to the LHS, reversing its sign.
        # If constant term is to the left of "=", we will take it to the RHS,␣
↪reversing its sign.

        # Going through the equation...
        while e[i] != "\\":
            coefValue = ""
            while e[i].isspace(): i = i + 1 # To traverse possible spaces␣
↪before '-'.
            if e[i] == "-": coefValue, i = coefValue + "-", i + 1 # Negative␣
↪sign detection.
            while e[i].isspace(): i = i + 1 # To traverse possible spaces after␣
↪'-'.

            # Number encountered...
            if e[i].isnumeric():
                coefValue, i = coefValue + e[i], i + 1
                while e[i].isnumeric() and e[i] != "\\":
                    coefValue, i = coefValue + e[i], i + 1
```

```
            # Alphabet encountered (potential variable)...
            if e[i].isalpha():
                varName, i = e[i], i + 1
                while e[i].isalnum() and e[i] != "\\":
                    varName, i = varName + e[i], i + 1
                # (This stores the entire unspaced string as a variable, if␣
↪encountered)

                # If variable already encountered in equation...
                if varName in varsFound:
                    coef[var[varName]][-1] = coef[var[varName]][-1] +␣
↪float(coefValue) * sign
                    # Coefficients get added.

                # If variable is newly encountered in the equation...
                else:
                    varsFound.append(varName)
                    # If the variable is newly encountered in the system...
                    if(varName not in var):
                        var[varName], varCount = varCount, varCount + 1
                        coef.append([])
                    # If no numerical coefficient specified...
                    if coefValue == "" or coefValue == "-": coefValue =␣
↪coefValue + "1"
                    # Making sure zero constants are put where required...
                    l = len(coef[var[varName]])
                    while l < eqCount:
                        coef[var[varName]].append(0)
                        l = l + 1
                    coef[var[varName]].append(float(coefValue) * sign)

            # If a constant is identified...
            elif coefValue != "":
                # If a constant already exists in the equation...
                if "c" in varsFound:
                    const[-1] = const[-1] + float(coefValue) * -sign
                # If a constant hasn't been encountered before...
                else:
                    varsFound.append("c")
                    const.append(float(coefValue) * -sign)

            # If equal-to sign encountered, invert the sign variable...
            else:
                if e[i] == "=": sign = -1
                i = i + 1
        eqCount = eqCount + 1
```

```
        # Making sure zero constant sums are put where required...
        if len(const) < eqCount: const.append(0)
    return (coef, const, var)

# Uses the lists of values from "getLists" and creates the necessary matrices...
def getMatrices(equationList):
    (coef, const, var) = getLists(equationList)
    nVar, nEq = len(coef), len(const)
    A = np.zeros((nEq, nVar))
    B = np.zeros((nEq, 1))
    for i in range(0, nEq):
        for j in range(0, nVar):
            try: A[i][j] = coef[j][i]
            except: A[i][j] = 0
    for i in range(0, nEq):
        B[i][0] = const[i]
    return (A, B, var)
```

Main function...

[15]:
```
# Uses the matrices from "getMatrices" and finds the solutions if they exists...

def solveSystem(equationList):
    (A, B, var) = getMatrices(equationList)
    A = np.matrix(A)
    B = np.matrix(B)
    print("Coefficient matrix:")
    print(A)
    print("Constant sum matrix:")
    print(B)
    try: np.linalg.inv(A)
    except:
        print("Inverse of coefficient matrix does not exist.")
        print("No solutions.")
    else: print("Inverse of coefficient matrix exists.")
    X = np.linalg.solve(A, B)
    for i in var:
        print("{0} = {1}".format(i, X[var[i], 0]))
```

1 EXAMPLES

1.1

[22]:
```
eqs = [ "8x + 5y = 9z",
        "3x + 2z = 9",
        "4y + 3z = 11x"]
solveSystem(eqs)
```

```
Coefficient matrix:
[[  8.   5.  -9.]
 [  3.   0.   2.]
 [-11.   4.   3.]]
Constant sum matrix:
[[0.]
 [9.]
 [0.]]
Inverse of coefficient matrix exists.
x = 1.4036697247706422
y = 2.064220183486239
z = 2.3944954128440368
```

1.2

[23]:
```
eqs = [ "x1 + 2x2 + x3 - 2x3 = -1",
        "2x1 + x2 + 4x3 = 2",
        "3x1 + 3x2 + 4x3 = 1"]
solveSystem(eqs)
```

```
Coefficient matrix:
[[ 1.  2. -1.]
 [ 2.  1.  4.]
 [ 3.  3.  4.]]
Constant sum matrix:
[[-1.]
 [ 2.]
 [ 1.]]
Inverse of coefficient matrix exists.
x1 = 1.6666666666666679
x2 = -1.3333333333333341
x3 = -3.7007434154171906e-16
```

2 CONCLUSION

This function allows for any arrangement of variables, any spacing, any variable names and any number of equations and variables, since its data structures are designed to dynamically generate their structure and data based on the inputs.