4. EQUATION SOLVING

f) Different approaches to solving systems of linear equations

November 27, 2021

1 AIM

We will try solving systems of linear differential equations using matrices, and using different approaches...

Support functions...

[2]:
```
# Imports and support functions for all the below operations...
from numpy import matrix, zeros, linalg
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...

[9]:
```
eq = ["3x + 6y - 4z = -3",
      "5 - 5z + y = x",
      "7x + 9y - 16z = 0"]
(A, B, var) = getMatrices(eq)
A = matrix(A)
B = matrix(B)
```

[10]:
```
# Method 1
print("\nMethod 1 solutions:")
X = linalg.inv(A)*B
for i in var:
    print("{0} = {1}".format(i, X[var[i], 0]))
# Method 2
print("\nMethod 2 solutions:")
X = linalg.solve(A, B)
for i in var:
    print("{0} = {1}".format(i, X[var[i], 0]))
# Method 2
print("\nMethod 2 solutions:")
X = A**(-1)*B
```

```
for i in var:
    print("{0} = {1}".format(i, X[var[i], 0]))
```

```
Method 1 solutions:
x = 2.4967741935483874
y = -1.6322580645161289
z = 0.17419354838709689

Method 2 solutions:
x = 2.496774193548387
y = -1.6322580645161289
z = 0.17419354838709686

Method 2 solutions:
x = 2.4967741935483874
y = -1.6322580645161289
z = 0.17419354838709689
```