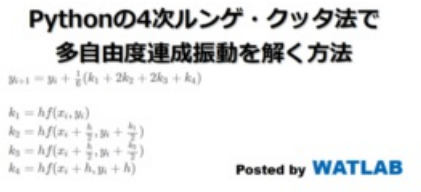
2020.02.17

Pythonの4次ルンゲ・クッタ法で多自由度連成振動を解く方法



多数の質点系から成る多自由度系は各点の振動が影響し合う連成振動をします。連成振動を解く方法は色々ありますが、ここでは有名な4次のルンゲ・クッタ法をPythonで作成して解いてみます。

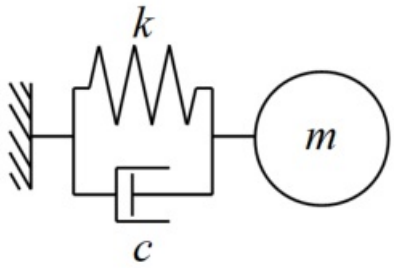
こんにちは。wat(@watlablog)です。ここでは**Pythonで作成したルンゲ・クッタコードで多自由度振動問題を解く方法を紹介します**！

目次（項目クリックでジャンプできます） [hide]

多自由度振動系の概要

1自由度のおさらい

多自由度振動系の話に行く前に、振動問題を理解するためには**1自由度振動系**の理解が必要になります。1自由度振動系のモデルは以下の図で表現されます。



当WATLABブログでは「Pythonで1自由度非減衰系の自由振動シミュレーション」より振動シミュレーション系の記事を書き始めました。まずはこちらの記事で1自由度の振動について理解しておくと、この後の話にスムーズに繋がるでしょう。

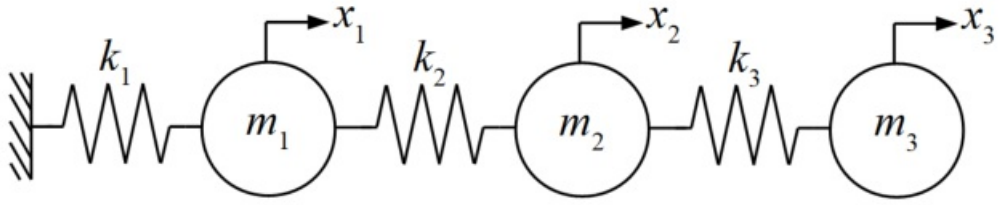
他にも関連記事として、
「Pythonで1自由度減衰系の自由振動シミュレーション」
「Pythonで1自由度減衰系の強制振動シミュレーション」
に1自由度系の記事を書いていますので、是非**減衰系**や**強制振動**の項目も参考にして頂ければと思います。

1自由度振動系のニュートンの運動方程式は式(1)となります。この式は強制振動形式で書いていますが、$_F$が0になれば自由振動となります。

$$m\ddot{x}+c\dot{x}+kx=F \quad (1)$$

多自由度振動系の運動方程式の立て方

多自由度振動系とは、自由度が複数の振動系です。以下の図は3自由度の非減衰系ですがこれも多自由度振動系と呼びます。



自由度が複数個に増えたと言っても、1自由度系と考え方は変わらず、このモデルの運動方程式は式(2)となります。

$$\mathbf{M}\ddot{\mathbf{x}}+\mathbf{K}\mathbf{x}=\mathbf{F} \quad (2)$$

減衰項も加えれば式(3)となります。

$$\mathbf{M}\ddot{\mathbf{x}}+\mathbf{C}\dot{\mathbf{x}}+\mathbf{K}\mathbf{x}=\mathbf{F} \quad (3)$$

各パラメータが太字になっていますが、これは行列形式で連立方程式を表現したに過ぎません（振動問題を検討する時、通常は自由度の数だけ運動方程式を必要とします）。

このように行列形式で運動方程式を立てた場合、各マトリクスの作り方は「Pythonで計算するために多自由度振動系を行列形式にする方法」に記載しましたので、もし不明な点があればこちらの記事を参考にしてみて下さい。

ルンゲ・クッタ法の概要

数値解析の必要性とルンゲ・クッタを選択する意味

1自由度の振動問題であれば微分方程式の一般解が存在しており、先に紹介した記事ではコードの検証として理論値との比較を行っていました。

しかし自由度が増え、多自由度になると解析的な値を求めることは困難を極め、通常は**数値解析**を行います。

先ほどの記事ではPythonの**scipy**というライブラリにある**odeint**というメソッドで数値解析を行っていました。

このページでもodeintを使ってサクッと簡単に解析をしようと思いましたが、**odeintの中身や引数の受け渡し等を多自由度に対応させる方法がちょっと調べただけではわかりませんでした**(お恥ずかしい)。

> 僕にとってブラックボックスであるodeintを紐解くよりも、既に世間で頻繁に使われているルンゲ・クッタ法を多自由度振動問題に対しnumpyで書いた方が早いと思いました。

…ということで、このページでは主にnumpyを使用して数値解析手法であるルンゲ・クッタ法コードを作成し、多自由度振動問題を数値的に解いていきます。

4次のルンゲ・クッタ法はプログラム的に大変書きやすく、精度や計算速度の関係もバランスの良い手法として知られています。まずはコードが書きやすいというのが、今回ルンゲ・クッタ法を選択した理由です。

ルンゲ・クッタ法(4次)の基本式

数値解析の世界では**オイラー法**(**Euler Method**)がシンプルな手法として有名ですが、精度が悪いことでも有名です。そして**ホイン法**(**Heun Method**)といったオイラー法よりも精度が改善された手法もありますが、次に紹介する4次のルンゲ・クッタ法ほどではありません。

ルンゲ・クッタ法(**Runge-Kutta Method**)は調べれば無数の説明ページが出てくるほど有名な手法で、問題の微分方程式をテイラー展開した時に、4次の項まで一致するという特徴を持ちます。

4次のルンゲ・クッタ法は式(4)で示され、以下に続く係数k_1,k_2,k_3,k_4を使います。

$$y_{i+1}=y_i+\frac{1}{6}(k_1+2k_2+2k_3+k_4) \quad (4)$$

$$k_1=hf(x_i,y_i)$$
$$k_2=hf(x_i+\frac{h}{2},y_i+\frac{k_1}{2})$$
$$k_3=hf(x_i+\frac{h}{2},y_i+\frac{k_2}{2})$$
$$k_4=hf(x_i+h,y_i+h)$$

ここでhは刻み幅として微小な値を設定します。

詳細な説明や導出はここでは省きますが（導出はかなり大変…）、今回はこの式を使って多自由度振動系の運動方程式を数値解析していきます。

> ルンゲ・クッタ法は他にも様々な改良が加えられた派生手法が存在しますが、ここでは一般に**古典的ルンゲ・クッタ法**と呼ばれる最もコーディングしやすい上式を使います。

4次のRunge-Kuttaで多自由度振動系を解析するコード

今回多自由度振動系の運動方程式をルンゲ・クッタ法で数値解析するにあたり、中々質量・減衰・剛性マトリクスを使った例題を探せませんでしたが、以下①②のWebページを大いに参考にさせて頂きました。

①パソコンで数値計算：ルンゲクッタ法
→複数方程式の扱い方を参考

②Qiita：[Pythonによる科学・技術計算]4次のルンゲ-クッタ法による1次元ニュートン方程式の解法
→Pythonのルンゲ・クッタ法記述スタイルを参考

ありがとうございました。

まずは1自由度振動系で確認

まずはルンゲ・クッタ法の確認として、これまでodeintでやってきたように1自由度振動系のシミュレーションを行い、理論値と比較をしてみます。

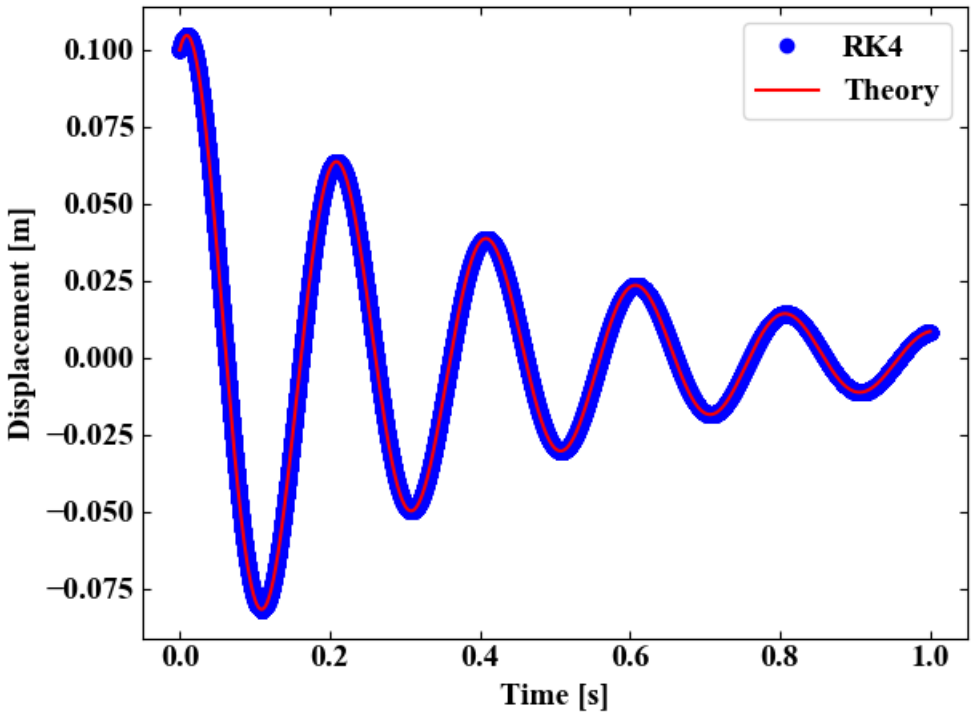
以下にコードを示しますが、運動方程式の変換は「Pythonで1自由度減衰系の自由振動シミュレーション」で説明した通りです。運動方程式とパラメータ値はdef関数の中に入れています。

式(4)のhはdtと時間刻みという意味合いで与えています。

```python
import numpy as np
import matplotlib.pyplot as plt

# □□□□□□□□□□□□□□□
def f(x, v):
    m = 1                           # □□[kg]
    c = 5                           # □□□□[N□(s/m)]
    k = 1000                        # □□[N/m]
    y = - (c/m) * v - (k/m) * x     # □□□□□□□□□□
    return y

# □□□□
t0 = 0.0                            # □□□□
t1 = 1.0                            # □□□□
dt = 0.00001                        # □□□□

# □□□□
x0 = 0.1                            # □□□□[m]
v0 = 1.0                            # □□□□[m/s]
x, v = x0, v0                       # □□□□□□

t_axis = np.arange(t0, t1, dt)      # □□□
x_sol = []                          # □□□x(□□□□□
v_sol = []                          # □□□v(□□)□□

# 4□□Runge-Kutta□□□□□□□□
for t in t_axis:
    v_sol.append(v)
    x_sol.append(x)
    k11 = f(x, v) * dt
    k12 = v * dt
    k21 = f(x + dt / 2, v + k11 / 2) * dt
    k22 = (v + k11 / 2) * dt
    k31 = f(x + dt / 2, v + k21 / 2) * dt
    k32 = (v + dt / 2) * dt
    k41 = f(x + dt, v + k31) * dt
    k42 = (v + k31) * dt
    v += (k11 + 2 * k21 + 2 * k31 + k41) / 6
    x += (k12 + 2 * k22 + 2 * k32 + k42) / 6

# □□□□□□□□□□
m = 1
c = 5
k = 1000
cc = 2 * np.sqrt(m * k)
zeta = c / cc
omega = np.sqrt(k / m)
omega_d = omega * np.sqrt(1 - np.power(zeta, 2))
sigma = omega_d * zeta
X = np.sqrt(np.power(x0, 2) + np.power((v0 + sigma * x0) / omega_d, 2))
phi = np.arctan((v0 + (sigma * x0)) / (x0 * omega_d))
theory = np.exp(- sigma * t_axis) * X * np.cos(omega_d * t_axis - phi)

# □□□□□□□□□□
# □□□□□□□□□□□□□□□□□□□
plt.rcParams['font.size'] = 14
plt.rcParams['font.family'] = 'Times New Roman'

# □□□□□□□□□□
plt.rcParams['xtick.direction'] = 'in'
plt.rcParams['ytick.direction'] = 'in'

# □□□□□□□□□□□□□□□□□□□
fig = plt.figure()
ax1 = fig.add_subplot(111)
ax1.yaxis.set_ticks_position('both')
ax1.xaxis.set_ticks_position('both')

# □□□□□□□□□□□□
ax1.set_xlabel('Time [s]')
ax1.set_ylabel('Displacement [m]')

# □□□□□□□□
ax1.plot(t_axis, x_sol, label='RK4', c='b', marker='o', linestyle='None')
ax1.plot(t_axis, theory, label='Theory', c='r')

fig.tight_layout()
plt.legend(loc='upper right')

# □□□□□□□□□□
plt.show()
plt.close()
```

変位と速度という複数方程式がありますが、先ほどの参考Webのようにそれぞれにルンゲ・クッタを適用しています。

このコードを実行すると以下の結果を得ます。
定量的な誤差は出していませんが、理論値と解析値は良く一致していると言って良い結果を得ました。



多自由度振動系に拡張

2自由度の場合の結果を固有値解析・参考書の例と比較してみる

1自由度で確認がとれたので、続いて多自由度で確認してみます。ここでは2自由度のシミュレーションコードを以下に示します。

def文の中で、行列同士の演算を単に掛け算記号「*」でやってしまうと、numpyの場合要素積になってしまうので、行列演算をするようにnp.dotを使っている所がポイントです。割り算は逆行列を同じくnp.dotでかけるという方法をとっています（順番にも注意）。

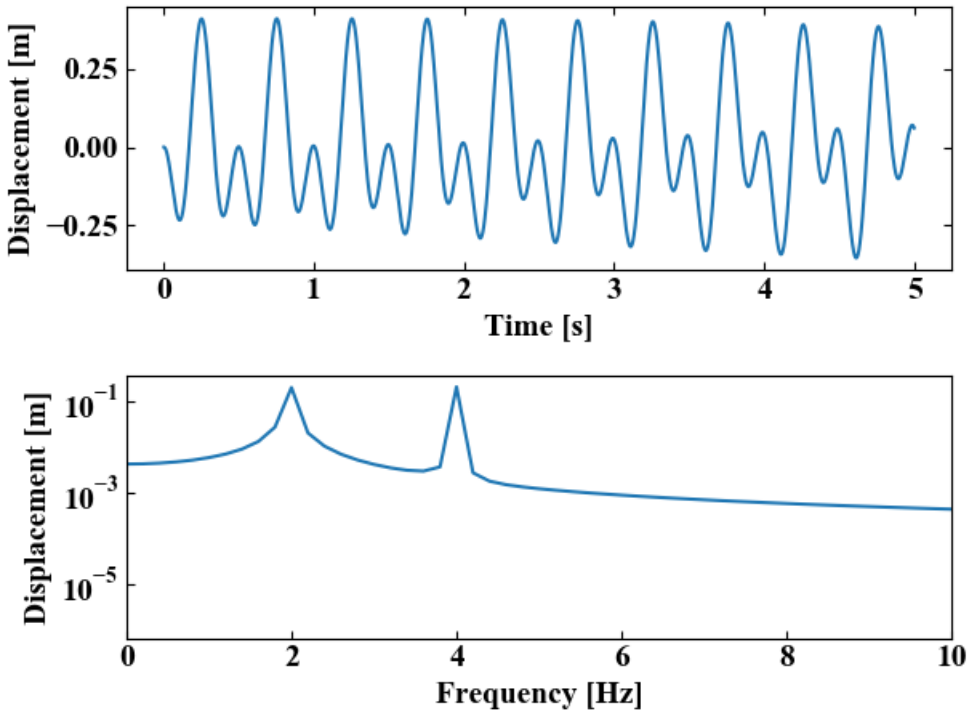
2自由度の場合は検証可能な時間波形の例題が見つかりませんでしたので、数値解析で得られた時間波形をフーリエ変換して固有振動数を確認する方法を選びました。

```
from scipy import fftpack
import numpy as np
import matplotlib.pyplot as plt

# □□□□□□□□□□□□□□□
def f(x, v):
    # □□□□□□□□□□□□□□□□□□□□
    m1 = 100
    m2 = 50
    k1 = 5.0e4
    k2 = 1.0e4

    M = np.array([[m1, 0],              # □□□□□□□□
                  [0, m2]])
    K = np.array([[k1 + k2, -k2],       # □□□□□□□□
                  [-k2, k2]])
    M_inv = np.linalg.inv(M)            # □□□□□□□□□□□□

    # □□□□□□□□□□□
    y = - np.dot((np.dot(K, M_inv)), x)
    return y

# □□□□□
t0 = 0.0                            # □□□□
t1 = 5.0                            # □□□□
dt = 1e-4                           # □□□□

# □□□□□
x0 = np.array([[-1.0], [0.0]])      # □□□□[m]
v0 = np.array([[0.0], [0.0]])       # □□□□[m/s]
x, v = x0, v0                       # □□□□□□□

t_axis = np.arange(t0, t1, dt)      # □□□
x_sol = []                          # □□□x(□□□□□
```

```
v_sol = []                          # □□□v(□□)□□

# 4□□Runge-Kutta□□□□□□□□
iteration = 0
for t in t_axis:
    x_sol.append(x[1, 0])           # 2□□□□□□□□□□□□□□□□□□□
    v_sol.append(v[1, 0])           # 2□□□□□□□□□□□□□□□□□□□
    print('iteration=', iteration,'time=', t)
    k11 = f(x, v) * dt
    k12 = v * dt
    k21 = f(x + dt / 2, v + k11 / 2) * dt
    k22 = (v + k11 / 2) * dt
    k31 = f(x + dt / 2, v + k21 / 2) * dt
    k32 = (v + k21 / 2) * dt
    k41 = f(x + dt, v + k31) * dt
    k42 = (v + k31) * dt
    v += (k11 + 2 * k21 + 2 * k31 + k41) / 6
    x += (k12 + 2 * k22 + 2 * k32 + k42) / 6
    iteration += 1

# □□□□□□□□□□
spec = fftpack.fft(x_sol)                       # □□□□□□□□□□□□□□□□□□□□□□□□
abs_spec = np.abs(spec / (len(spec)/2))          # □□□□□
frequency = np.linspace(0, 1/dt, len(x_sol))     # □□□□□□□□

# □□□□□□□□□□
# □□□□□□□□□□□□□□□□□□□
plt.rcParams['font.size'] = 14
plt.rcParams['font.family'] = 'Times New Roman'

# □□□□□□□□□□
plt.rcParams['xtick.direction'] = 'in'
plt.rcParams['ytick.direction'] = 'in'

# □□□□□□□□□□□□□□□□□□□
fig = plt.figure()
ax1 = fig.add_subplot(211)
ax1.yaxis.set_ticks_position('both')
ax1.xaxis.set_ticks_position('both')
ax2 = fig.add_subplot(212)
ax1.yaxis.set_ticks_position('both')
ax1.xaxis.set_ticks_position('both')

# □□□□□□□□□□□□□
ax1.set_xlabel('Time [s]')
ax1.set_ylabel('Displacement [m]')
ax2.set_xlabel('Frequency [Hz]')
ax2.set_ylabel('Displacement [m]')

ax2.set_xlim(0, 10)
ax2.set_yscale('log')

# □□□□□□□□
ax1.plot(t_axis, x_sol, label='RK4')
ax2.plot(frequency, abs_spec)

fig.tight_layout()

# □□□□□□□□□□
plt.show()
plt.close()
```

上記コードを実行すると以下の結果を得ます。
時間波形には2つの質点の振動が相互に影響している様子が現れ、周波数波形にはピークが2つ発生していました。

今回は自由振動のモデルで計算を行っているので、これらのピークは固有振動数が近似されているはずです。



このモデルの固有振動数が本当にこの2つのピークなのか、検証のために以下のコードで固有値解析を行ってみます。

```python
import numpy as np
from matplotlib import pyplot as plt

# □□□□□□□□□
m1 = 100
m2 = 50
k1 = 5.0e4
k2 = 1.0e4

# □□□□□□□□□□□□□□□□□□□□
M = np.array([[m1, 0],
              [0, m2]])
K = np.array([[k1 + k2, -k2],
              [-k2, k2]])

# □□□□□□□□□□□□□□□
M_inv = np.linalg.inv(M)

# □□□□□□□□□□□□□□
omega, v = np.linalg.eig(np.dot(M_inv, K))

# □□□□□□□□□□□□□□
omega_sort = np.sort(omega)

# □□□□□□□□□□□□□□□□□□□□
# □□□□□□□□□□□□□□□□
sort_index = np.argsort(omega)

# □□□□□□□□□□□□□□□□□□□□
v_sort = []
for i in range(len(sort_index)):
    v_sort.append(v.T[sort_index[i]])
v_sort = np.array(v_sort)

# □□□□□□□□□□□□
print(np.sqrt(omega_sort) /(2 * np.pi))
```

ちなみに、固有値解析のコードは「Pythonで多自由度系の固有値解析!固有振動数とモードを計算」で書いたものを使用しています（最後の固有角振動数を固有振動数として周波数に変換している所が異なる程度）。

この記事には固有ベクトル（モードシェイプ）の確認方法等も記載していますので、是非今回の数値解析と併用してみて下さい。

上記固有値解析のコードを実行すると、以下の結果を得ます。
固有振動数は2つ持ち、それぞれ約2[Hz], 4[Hz]と、先ほどのグラフと対応していることがわかりました。

```
[1.98178918 4.04187436] # □□□□□[Hz]
```

そして最後はダメ押しです。今回使用した質量と剛性のパラメータは「**モード解析入門**」という参考書の**P130、3.4数値例**に記載されたものです。

モード解析入門　長松昭男著
「わかりやすい」「すぐ役に立つ」「なぜ、から始める」「数学に頼らない」「数学的厳密さを保つ」「多様な読者に対応できる」を意識して書かれた良書で、振動解析従事者必携の本と思います。

Amazon　楽天

参考書の結果も1.98[Hz], 4.04[Hz]となっているので、周波数は数値解析、固有値解析、参考書で矛盾はありませんでした。

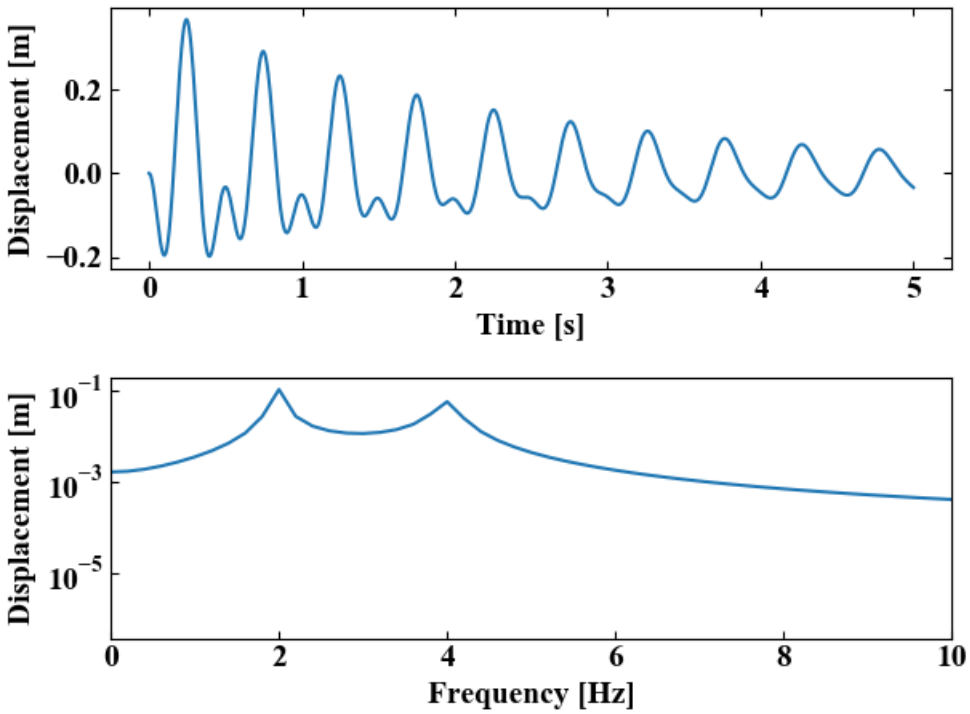
初めての多自由度系の数値解析プログラムだったので正直不安でしたが、どうやら固有振動数レベルでは問題無いように見えます。

減衰マトリクスを追加してみる

続いてさらに汎用性を高めるために、減衰係数[N・(s/m)]を行列形式にした**減衰マトリクス**を追加してみます。多自由度振動系ルンゲ・クッタコード内def関数の中の定数とマトリクス、モデルの式$_y$を変更しました。

```
# □□□□□□□□□□□□□□□
def f(x, v):
    # □□□□□□□□□□□□□□□□□□□□
    m1 = 100
    m2 = 50
    c1 = 50
    c2 = 50
    k1 = 5.0e4
    k2 = 1.0e4

    M = np.array([[m1, 0],              # □□□□□□□□
                  [0, m2]])
    C = np.array([[c1 + c2, -c2],       # □□□□□□□□
                  [-c2, c2]])
    K = np.array([[k1 + k2, -k2],       # □□□□□□□□
                  [-k2, k2]])
    M_inv = np.linalg.inv(M)            # □□□□□□□□□□□□
    
    # □□□□□□□□□□
    y = - np.dot((np.dot(C, M_inv)), v) - np.dot((np.dot(K, M_inv)), x)
    return y
```

時間波形は徐々に振動が小さくなっていく様子が、そして周波数波形はピークが鈍って減衰の影響が反映されたことがわかりました。

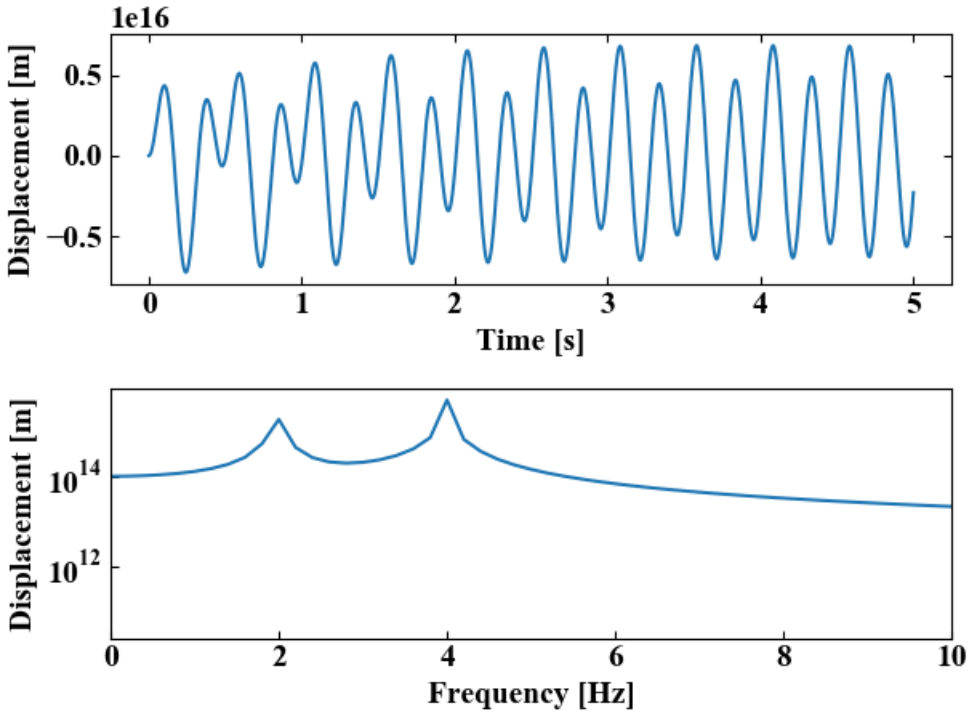


強制振動モデルとして外力をかけてみる

最後に以下のコードで外力ベクトルを設定、モデルにも外力項を追加して強制振動モデルにしてみました。先ほどと同じくモデルの変更はdef文の中を変更するだけです。

今回は単なる外力が反映されるかどうかの確認なので、トンデモ力を4[Hz]と共振にぶちあてるような感じで与えています。

```python
# □□□□□□□□□□□□□□□
def f(x, v):
    # □□□□□□□□□□□□□□□□□□□□□
    m1 = 100
    m2 = 50
    c1 = 50
    c2 = 50
    k1 = 5.0e4
    k2 = 1.0e4

    M = np.array([[m1, 0],              # □□□□□□□□
                  [0, m2]])
    C = np.array([[c1 + c2, -c2],       # □□□□□□□□
                  [-c2, c2]])
    K = np.array([[k1 + k2, -k2],       # □□□□□□□□
                  [-k2, k2]])
    M_inv = np.linalg.inv(M)            # □□□□□□□□□□□□

    # □□□□□□□
    F = np.array([[0.0], [1.0e20 * np.cos(2 * np.pi * 4.0 * t)]])

    # □□□□□□□□□□
    y = np.dot(M_inv, F) - np.dot((np.dot(C, M_inv)), v) - np.dot((np.dot(K, M_inv)), x)
    return y
```



…やりすぎました。これは地球がやばい。
以下がコード実行結果ですが、プロットの縦軸がトンデモ無い値になっています。そして4[Hz]のピークが第一ピークよりも高くなっています。計算の目的からすると、外力が働いていることが確かめられたので良しとします。

自由度を増やす場合は上記コードの質量・減衰・剛性に関する正方マトリクスや、初期変位ベクトル、初期速度ベクトル、外力ベクトルをそれぞれ増やしていくだけです。

まとめ

本記事は1自由度振動系のおさらいからはじめ、多自由度振動系の問題を4次のルンゲ・クッタ法で数値計算してみました。

1自由度は理論値と、2自由度は固有振動モードと比較しながらPythonコードを作り、非減衰、減衰、強制振動問題で動作を確認しました。

ただやってみた感想ですが、普通のノートPCだと計算が遅いと感じます。
（for文で回しているから？）

もっと効率の良いコードが書ければ良いと思いますが、そもそも速度を重視するのであれば言語の選定も重要ではと感じました。

数値解析はC言語やC++、java、FORTRANが高速計算に向いているとの噂も。
しかし、Pythonによるシミュレーションは機械学習やその他多数の便利ライブラリとの合わせ技を行う場合にメリットがありそうです。

ルンゲ・クッタ法を初めてコーディングしてみました！数値計算は色々な方程式の形を見ることができるので楽しいですね！
Twitterでも関連情報をつぶやいているので、wat(@watlablog)のフォローお待ちしています！

シミュレーション シミュレーション, ルンゲ・クッタ, 多自由度, 強制振動, 振動, 数値解析

wat

機械工学を専攻し大学院を修了後、 技術系の職に就き日々実験やシミュレーションを使う仕事をしています。 このブログでは初心者が科学技術プログラムを作れるようになることを目標に、学習結果を記録していきます。

コメント

mariomario より:2020年2月24日 10:10 PM

先日、コメントさせて頂いたものです。Pythonによる物理シュミレーションの初心者である私に、ここまで丁寧に対応していただきまして、感激しております。
面白かったのは、ばね定数”k1″ を大きくすると質点1の振動数も増えておりますが、質点”２”の振動数も大きくなっていることがきちんと描写できている点でした。
当たり前かもしれませんが、質点2に繋がっていないバネ１のバネ定数を変えると、質点2の振動数も変化することが描写出来ていて、改め数値計算の面白さに触れることが出来ました。
(念のため、質点１の変位と速度を抽出するために以下にさせて頂きました)
x_sol.append(x[0, 0]) # 1つ目の自由度の変位を結果として抽出
v_sol.append(v[0, 0]) # 1つ目の自由度の速度を結果として抽出
引き続き、wat様のブログで勉強させていただきたく思います。
今後とも、どうぞよろしくお願いいたします。

返信

wat より:2020年2月24日 10:45 PM

ご丁寧に返信誠にありがとうございます！
こちらもまだまだ初心者でして、これを機に学習することができ質問して頂いたことに感謝しています。
僕も初心者であるためおそらく同じ所で面白さを感じていると思います。
今回の解法ができるようになったので、次は制振に使われる2自由度の動吸振器理論もシミュレーションしてみたいと思いました。
今後ともよろしくお願い致します。

返信

コメントを残す

名前 *

メールアドレス（公開はされません。） *

次回のコメントで使用するためブラウザーに自分の名前、メールアドレス、サイトを保存する。

コメントを送信

最近の投稿

- 勾配降下法に慣性項を追加するMomentumをPythonで実装
- 勾配降下法を回帰分析に適用する式と実装のためのPythonコード
- Pythonで1変数と2変数関数の勾配降下法を実装してみた
- 多次元解析チャートで3個以上の多変量グリッドサーチ結果を可視化

アーカイブ

カテゴリー

- シミュレーション (9)
- 信号処理 (31)
- 数学 (5)
- 書籍 (1)
- 画像処理 (26)