Numerical Analysis, Basic Course: Project Assignement

Reuben Lindroos

The following pendulum equation:

$$\frac{\delta^2 \alpha}{\delta t^2} = -\frac{g}{l} sin \alpha(t)$$

Where:

$$\omega = \frac{\delta \alpha}{\delta t}$$

$$\frac{\delta \omega}{\delta t} = \frac{\delta^2 \alpha}{\delta t^2} = \frac{-g}{l} sin(\alpha(t))$$

Has the non linear solution (using the Trapezoidal Rule):

$$\begin{pmatrix} \alpha_{n+1} \\ \omega_{n+1} \end{pmatrix} = \begin{pmatrix} \alpha_n \\ \omega_n \end{pmatrix} + \frac{h}{2} \left(\begin{pmatrix} \omega_n \\ \frac{-g}{l} sin(\alpha_n) \end{pmatrix} + \begin{pmatrix} \omega_{n+1} \\ \frac{-g}{l} sin(\alpha_{n+1}) \end{pmatrix} \right)$$

If we denote a new function **F** by

$$\mathbf{F} \begin{pmatrix} \alpha_{n+1} \\ \omega_{n+1} \end{pmatrix} = 0$$

where

$$\mathbf{F} \begin{pmatrix} \alpha \\ \omega \end{pmatrix} = \begin{cases} \alpha - \frac{h}{2}\omega - \alpha_n - \frac{h}{2}\omega_n \\ \omega + \frac{h}{2}\frac{g}{l} sin(\alpha) - \omega_n + \frac{h}{2}\frac{g}{l} sin(\alpha_n) \end{cases}$$

This function can then be used to for a fixed point iteration of the intervals of the integral. This fixed point is placed at the value:

$$\mathbf{F} \begin{pmatrix} \alpha \\ \omega \end{pmatrix} = 0.$$

So for every fixed point iteration, we will see a convergence of the function to zero after a set number of iterations (provided the laws for our function and interval are satisfied). This therefore is a method of quantifying the error by which our trapezoidal method plots, as we can see if the guess is too far away, or outside of the interval, we will see a very slow convergence or no convergence whilst if our guess is in the appropriate value we will have a fast convergence. This will depend upon our initial guess and the definition for our function. The function has to be Lipschitz continuous over the interval and has to have well defined iterates. The code on the following page was used to show this.

```
def fpir(h, alphainit ,alphanew ,omegainit ,omeganew ,n):
    alpha=alphanew
    omega=omeganew
    g=-9.81
    l=1
    v=array([alpha ,omega])

    for i in range(1,n):
        alphanew=alpha-(h/2)*omega-alphainit -(h/2)*omegainit
        omeganew=omega+(h/2)*(g/l)*sin(alpha)-omegainit+(h/2)*(g/l)*sin(alphainit)
        if abs(omega-omeganew)<=1.e-3:
            print (omega, alpha)
        alpha=alphanew
        omega=omeganew
        omegainit=omeganew
        alphainit=alphanew
        v=vstack((v,(alphanew ,omeganew)))
        i+=1
    return v
```

This code showed a strong convergence with the guesses from our numerical method (shown on the next page). After only a few iterations we were already near zero. To apply Newtons method to this system the Jacobian of the vector function has to be found:

$$\mathbf{J}_F = \begin{pmatrix} \frac{\delta F_\alpha}{\delta\alpha} & \frac{\delta F_\alpha}{\delta\omega} \\ \frac{\delta F_\omega}{\delta\alpha} & \frac{\delta F_\omega}{\delta\omega} \end{pmatrix} = \begin{pmatrix} 1 & -1 \\ \frac{g}{l} cos\alpha & 1 \end{pmatrix}$$

and therefore Newtons method for this iteration would be:

$$\begin{pmatrix} \alpha_n^{(n+1)} \\ \omega_n^{(n+1)} \end{pmatrix} = \begin{pmatrix} \alpha_n^{(n)} \\ \omega_n^{(n)} \end{pmatrix} - \begin{pmatrix} 1 & -1 \\ \frac{g}{l} cos\alpha & 1 \end{pmatrix}^{-1} \mathbf{F} \begin{pmatrix} \alpha \\ \omega \end{pmatrix}$$

All though Newtons method does have the difficulty of having to compute a Jacobian as well as its inverse. In our case is not a disaster however if it were a larger system this could be unfeasible. Newtons Method, however, does have the advantage of being locally quadratic convergent under certain conditions. This in turn would mean that the evaluation of our guess would be more precise should we decide to use this method.

The numerical method used to plot this function was the Trapezoidal Interpolation method. We therefore assign the values:

$$k1a = \omega_n h \qquad k2a = h(\omega_n + hg(\alpha(t))$$

$$k1b = hg(\alpha(t)) \qquad k2b = hg(\alpha(t) + \omega_n h)$$

for

$$g(\alpha(t)) = \frac{-g}{l} sin(\alpha(t))$$

Now we can see we have an expression for α_{n+1} and ω_{n+1}:

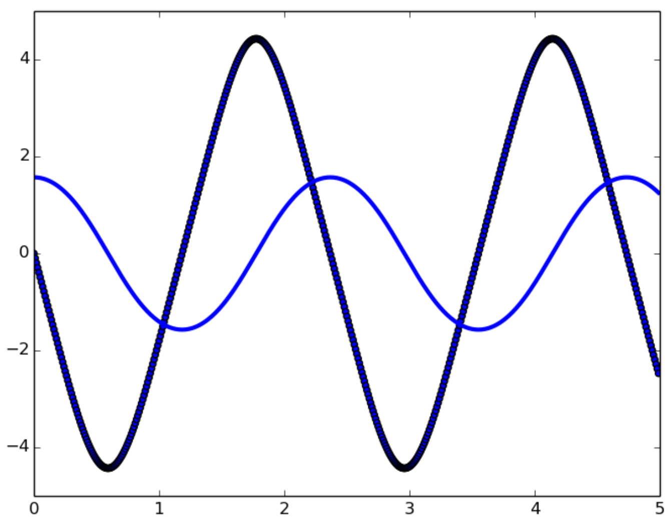
$$\alpha_{n+1} = \alpha_n + \frac{k1a + k2a}{2}$$

$$\omega_{n+1} = \omega_n + \frac{k1b + k2b}{2}$$

The algorithm to carry these operations out are as follows:

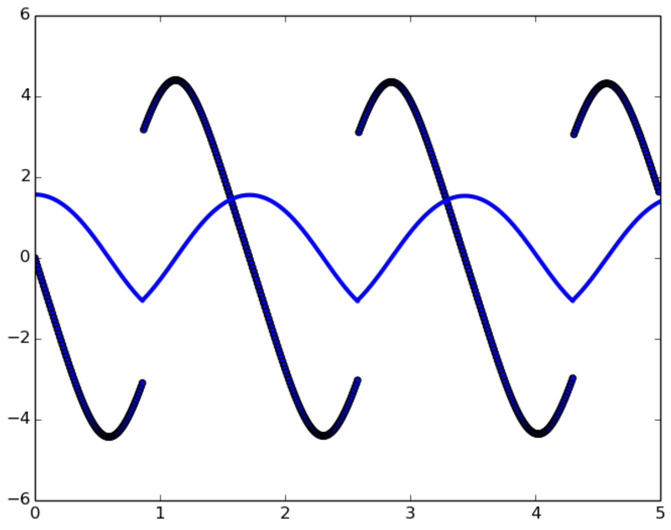
```
def trapfunc(h,alpha,omega,n,t):
    v=array([alpha,omega,t])
    l=[]
    l1=[]
    l2=[]
    for i in range(1,n):
        k1a=h*omega
        k1b=h*g(alpha)
        k2a=h*(omega+k1b)
        k2b=h*g(alpha+k1a)
        alpha+=(k1a+k2a)/2
        omega+=(k1b+k2b)/2
        t+=h
        l.append(alpha)
        l1.append(omega)
        l2.append(t)
        v=vstack((v,(alpha,omega,t)))
        i+=1
    return v,l,l1,l2
```

The first tuple index of this code will return a 3 column array with the values of α, ω and t respectively.



The figure shows the plot of $\alpha(t)$ and $\omega(t)$(dotted) plotted over n iterations for h =0.01, with initial values of $g(\alpha(0)) = -g/l$, $\alpha(0) = \pi/2$ and $\omega(0) = 0$.

To solve for the collision with the obstacle, the Lagrange method for interpolation polynomials was used. The nodes t_{n-1}, t_n and t_{n+1} were used to "extrapolate" $\alpha(t)$. Solving for this polynomial we get the following results:



The figure shows the plot as before with $\alpha_{obst} = \frac{2\pi}{3}$ and setting $\omega_{obst} = -\omega(t_n)$ for every re-initialization of the iteration.

The following if statement was added to the code to achieve this:

```
falpha=lgrngpoly(array([t1,t2,t3]),array([a1,a2,a3]))
    if falpha(t)<=alphaobst:
        omega=-omega
        alpha=alphaobst
```

And likewise for ω.