AMATH 585 Assignment 3

Tyler Chen

Problem 1 (nonlinear pendulum)

(a) Write a program to solve the boundary value problem for the nonlinear pendulum as discussed in the text. See if you can find yet another solution for the boundary conditions illustrated in Figures 2.4 and 2.5.

(b) Find a numerical solution to this BVP with the same general behavior as seen in Figure 2.5 for the case of a longer time interval, say, $T = 20$, again with $\alpha = \beta = 0.7$. Try larger values of T. What does $\max_i \theta_i$ approach as T is increased? Note that for large T this solution exhibits "boundary layers".

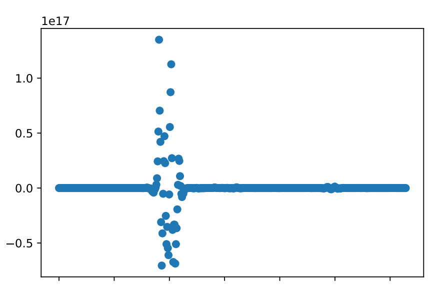
Solution

(a) We implement Newton's method to solve the system outlined in the book.

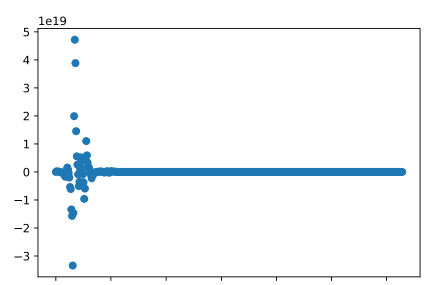
```
# problem discritization
def G(theta):
    Gout = np.zeros(m+1)
    for i in range(1,m):
        Gout[i] = (theta[i-1]-2*theta[i]+theta[i+1])/h2+np.sin(theta[i
            ])
    return Gout[1:m+1] # return only inner things since boundaries are
        fixed

def J(theta):
    Jout = np.triu(np.tril(np.ones((m,m)),1),-1)-np.identity(m) #
        trigiagonal all ones
    Jout += np.diag(-2 + h2*np.cos(theta[1:m+1]))
    return Jout/h2

# problem parameters
alpha = 0.7
beta = 0.7
T = 2*np.pi # part a
x = np.linspace(0,T,m+2)
h = T/(m+1)
h2 = h**2

# discritization parameters
m = 512
x = np.linspace(0,T,m+2)

#initial guess
theta = 0.7*np.cos(x)+0.5*np.sin(x)
for k in range(25):
        print(k)
        delta = np.linalg.solve(J(theta),-G(theta))
        theta[1:m+1] += delta

        if max(abs(delta)) < 10e-14:
            break

plt.figure()
plt.scatter(x,theta)
plt.savefig('img/1/original.pdf')
```

```
theta = 0.7 + abs(x-np.pi)-np.pi
for k in range(25):
        print(k)
        delta = np.linalg.solve(J(theta),-G(theta))
        theta[1:m+1] += delta

        if max(abs(delta)) < 10e-14:
            break

plt.figure()
plt.scatter(x,theta)
plt.savefig('img/1/abs.pdf')

maxtheta = []
theta = 0.7 + np.sin(x/2)
for T in np.linspace(6,62,8):

    x = np.linspace(0,T,m+2)
    h = T/(m+1)
    h2 = h**2

    # Newton's method
    for k in range(25):
        print(k)
        delta = np.linalg.solve(J(theta),-G(theta))
        theta[1:m+1] += delta

        if max(abs(delta)) < 10e-14:
            break

    maxtheta = np.append(maxtheta,max(abs(theta)))
    plt.figure()
    plt.scatter(x,theta)
    plt.savefig('img/1/'+str(int(T))+'.pdf')

print(maxtheta)
```

Figure 1a shows the solution converging to the solution found in the book. Figure 1b shows a solution not found in the book. Physically this corresponds to sending the pendulum clockwise towards the top, and then having it fall back down. Similar to book Figure 2.5, but in the opposite direction.

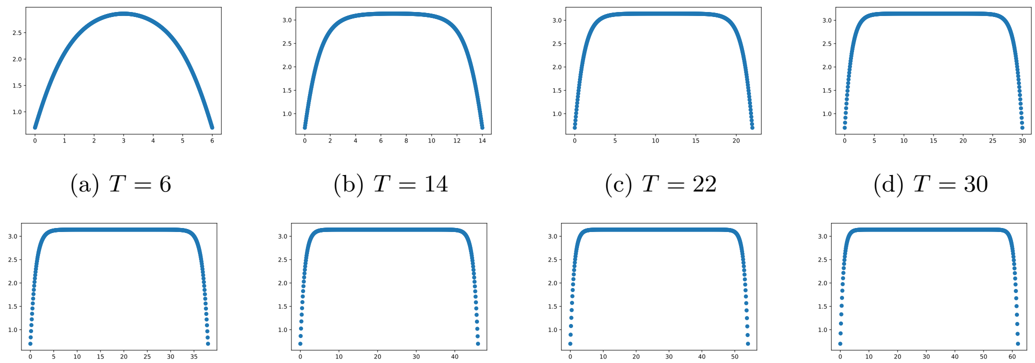


(a) $\theta^{[0]} = 0.7\cos(x) + 0.7\sin(x)$



(b) $\theta^{[0]} = 0.7 + |x - \pi| - \pi$

(b) We now increase T. In order to find a convergent solution we use the previously found convergent solution, scaled (by just using the same indices) from a slightly slower value of T. The results are show in Figure 2.



(a) $T = 6$ (b) $T = 14$ (c) $T = 22$ (d) $T = 30$

(e) $T = 38$ (f) $T = 46$ (g) $T = 54$ (h) $T = 62$

Figure 2: Plots of solution for varying T

The largest value of theta seems to converge to π as seen in Table 1.

T	$\max_i \lvert\theta_i\rvert$
6	2.8598081466662055
14	3.1364877262959778
22	3.141499073564805
30	3.1415909367890777
38	3.1415926220614177
46	3.141592653010052
54	3.1415926535791168
62	3.1415926535895964

Table 1: largest value of θ for a given T

Physically this makes sense. In order for the pendulum to be at $\theta = 0.7$ at time 0 and T, with the pendulum moving up and counterclockwise at time 0, it must go up and almost balance for some time before falling back. If we increase T, the time it spends at the top must be longer and longer. That is, the pendulum will almost be vertical, so that the acceleration is almost zero since it is perpendicular to the force of gravity.

Problem 2 (Gershgorin's theorem and stability)

Consider the boundary value problem

$$-u_{xx} + (1+x^2)u = f, \quad 0 \le x \le 1,$$

$$u(0) = 0, \quad u(1) = 0.$$

On a uniform grid with spacing $h = 1/(m+1)$, the following set of difference equations has local truncation error $O(h^2)$:

$$\frac{2u_i - u_{i+1} - u_{i-1}}{h^2} + (1+x_i^2)u_i = f(x_i), \quad i = 1, \ldots, m.$$

(a) Use Gerschgorin's theorem to determine upper and lower bounds on the eigenvalues of the coefficient matrix for this set of difference equations.

(b) Show that the L_2-norm of the *global error* is of the same order as the local truncation error.

Solution

(a) We apply the Gershgorin Theorem for rows.

Fix $i = 2, \ldots, m-1$ and let A denote the coefficient matrix for this set of difference equations (note A depends on h).

Then A is tridiagonal symmetric with,

$$a_{ii} = 2/h^2 + (1+x_i^2) \qquad a_{i,i-1} = a_{i,i+1} = -1/h^2$$

Thus, the Gershorin row disk have radii,

$$r_i = \sum_{j \ne i} |a_{ij}| = |-1/h^2| + |-1/h^2| = 2/h^2$$

Since A is real and symmetric, all eignevalues are real. Denote the part of the i-th row disk on the real axis by D_i. Then,

$$D_i = [a_{ii} - r_i, a_{ii} + r_i] = [(1+x_i^2), 4/h^2 + (1+x_i^2)]$$

Given $h = 1/(m+1)$ and $x_i = ih$ we have,

$$D_i = [1 + i^2/(m+1)^2, 4(m+1)^2 + 1 + i^2/(m+1)^2]$$

We also have,

$$r_1 = |1/h^2| \qquad r_m = |1/h^2|$$

So that,

$$D_1 = [(m+1)^2 + 1 + 1/(m+1)^2, 3(m+1)^2 + 1 + 1/(m+1)^2]$$
$$D_m = [(m+1)^2 + 1 + m^2(m+1)^2, 3(m+1)^2 + 1 + m^2/(m+1)^2]$$

For reasonable values of m the disks overlap. However, all eigenvalues are contained in $\cap_i D_i$. That is if λ is an eigenvalue of A,

$$1 + 2^2/(m+1)^2 \leq \lambda \leq 4(m+1)^2 + 1 + (m-1)^2/(m+1)^2$$

(b) From above it is obvious that all eigenvalues of A are larger than one. Then, the eigenvalues of A^{-1} are all less than or equal to one.

Thus, this difference method is stable as it is linear, and if h is sufficiently small,

$$\left\|A^{-1}\right\|_2 = \rho(A^{-1}) < 1$$

Since the local truncation error is $\mathcal{O}(h^2)$ and the method is stable, the global error is also $\mathcal{O}(h^2)$. $\square$

Problem 3 (Richardson extrapolation)

Use your code from problem 6 in assignment 1, or download the code from the course web page to do the following exercise. Run the code with $h = .1$ (10 subintervals) and with $h = .05$ (20 subintervals) and apply Richardson extrapolation to obtain more accurate solution values on the coarser grid. Record the L_2-norm or the ∞-norm of the error in the approximation obtained with each h value and in that obtained with extrapolation.

Suppose you assume that the coarse grid approximation is piecewise linear, so that the approximation at the midpoint of each subinterval is the average of the values at the two endpoints. Can one use Richardson extrapolation with the fine grid approximation and these interpolated values on the coarse grid to obtain a more accurate approximation at these points? Explain why or why not?

Solution

We run the code from assignment one with $m = 10$ and $m = 20$ outputting to a 2×20 array. We then apply Richardson extrapolation as,

```
# richardson extrapolation
m = 10
h = 1/m
x = np.linspace(0,1,m+1)
richardson = (4*res[1,1::2] - res[0,0:10])/3

plt.scatter(x[1:],((1-x[1:])**2)) #actual solution
plt.scatter(x[1:],richardson) # numerical solution
print("m=10:",np.sqrt(h)*np.linalg.norm(res[0,0:10]-(1-x[1:])**2))
print("m=20 (subsampled):",np.sqrt(h)*np.linalg.norm(res[1,1::2]-(1-x[1:])
    **2))
print("richardson:",np.sqrt(h)*np.linalg.norm(richardson-(1-x[1:])**2))
```

The L_2 norm of the code with $m = 10$, $m = 20$ subsampled down to 10 samples, and the Richardson extrapolation are shown in Table 2

method	L_2 error
$m = 10$	0.0306517881736
$m = 20$	0.0074474900883
richardson	0.000290117131706

Table 2: L_2 errors using various methods

Richardson extrapolation works by cancelling the first nonzero coefficient in the error. That is, given an order h^k approximation,

$$\tilde{u}(x,h) = u(x) + a_k(x)h^k + \mathcal{O}(h^{k+1})$$

by taking our new approximation as

$$\frac{2^k \tilde{u}(x,h/2) - u(x,h)}{2^k - 1} = u(x) + \mathcal{O}(h^{k+1})$$

we gain one order of accuracy.

However, this relies on the constants in the expansions for a given x value to be the same.

Suppose the fine grid spacing is h and consider a point x on the finer grid between two points $x-h$ and $x+h$ on the coarse grid. The linear interpolation is then,

$$\begin{aligned}\frac{\tilde{u}(x-h)+\tilde{u}(x+h)}{2} &= \frac{u(x-h)+a(x-h)h^2+u(x+h)+a(x+h)h^2}{2} \\ &= u(x)-\frac{h^2}{2}\frac{2u(x)-u(x-h)-u(x+h)}{h^2}+h^2\frac{a(x-h)+a(x+h)}{2} \\ &= u(x)-h^2\frac{a(x-h)+a(x+h)-u''(x)}{2}\end{aligned}$$

Thus, unless $a(x)=(a(x-h)+a(x+h))/2-u''(x))/2$ we cannot use richardson extrapolation in the normal way (with $h, h/2$ and coefficients 4 , -1 and dividing by 3).

Problem 4

Write down the Jacobian matrix associated with Example 2.2 and the nonlinear difference equations (2.106) on p. 49. Write a code to solve these difference equations when $a = 0$, $b = 1$, $\alpha = -1$, $\beta = 1.5$, and $\epsilon = 0.01$. Use an initial guess of the sort suggested in the text. Try, say, $h = 1/20$, $h = 1/40$, $h = 1/80$, and $h = 1/160$, and turn in a plot of your results.

Solution

We implement Newton's method to solve $G(U) = 0$ where,

$$G_i(U) = \epsilon\left(\frac{U_{i-1} - 2U_i + U_{i+1}}{h^2}\right) + U_i\left(\frac{U_{i+1} - U_{i-1}}{2h} - 1\right)$$

We compute the Jacobian as,

$$J_{ij}(U) = \begin{cases} \epsilon/h^2 - U_i/2h & j = i - 1 \\ -2\epsilon/h^2 + (U_{i+1} - U_{i-1})/2h - 1 & j = 1 \\ \epsilon/h^2 + U_i/2h & j = i + 1 \\ 0 & \text{otherwise} \end{cases}$$

At each step we solve,

$$J(U^{[k]})\delta^{[k]} = -G(U^{[k]})$$

starting with initial guess,

$$U^{[0]} = x - \bar{x} + w_0 \tanh(w_o(x - \bar{x})/2\epsilon)$$

where,

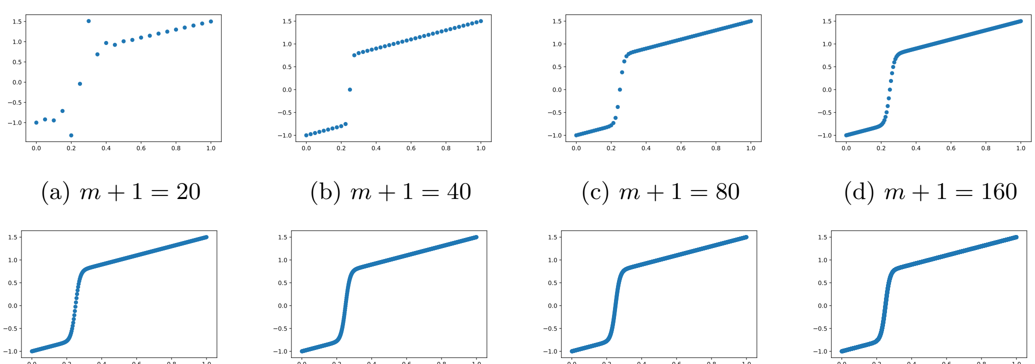
$$\bar{x} = \frac{1}{2}(a + b - \alpha - \beta) \qquad\qquad w_0 = \frac{1}{2}(a - b + \beta - \alpha)$$

We choose the terminating condition $\|\delta\|_\infty < 10^{-14}$ or $k = 25$ iterations. This is implemented in Python

```
# problem discritization
def G(U):
    Gout = np.zeros(m+2)
    for i in range(1,m):
        Gout[i] = eps*(U[i-1]-2*U[i]+U[i+1])/h2+U[i]*((U[i+1]-U[i-1])/(2*h)
            -1)
    return Gout[1:m+1]

def J(U):
    Jout = np.zeros((m+2,m+2))
    for i in range(1,m+1):
        Jout[i,i-1] = eps/h2-2*U[i]/(2*h)
        Jout[i,i] = -2*eps/h2+(U[i+1]-U[i-1])/(2*h)-1
        Jout[i,i+1] = eps/h2+2*U[i]/(2*h)
    return Jout[1:m+1,1:m+1]

# problem parameters
```

```
alpha = -1
beta = 1.5
eps = 0.01

a = 0
b = 1

# discritization parameters
for e in [1,2,3,4,5,6,8]:
    m = 10*2**e-1
    x = np.linspace(a,b,m+2)
    h = (b-a)/(m+1)
    h2 = h**2

    # initial guess
    xb = (a+b-alpha-beta)/2
    w0 = (a-b+beta-alpha)/2
    U = x - xb + w0*np.tanh(w0*(x-xb)/(2*eps))

    # Newton's method
    for k in range(25):
        print(k)
        delta = np.linalg.solve(J(U),-G(U))
        U[1:m+1] += delta

        print(max(abs(delta)))
        if max(abs(delta)) < 10e-14:
            break

    plt.figure()
    plt.scatter(x,U)
    plt.savefig('img/4/'+str(m+1)+'.pdf')
```

We run the code for $m = 10 \times 2^n - 1$ for $n \in \{1, 2, 3, 4, 5, 6, 7, 10\}$. The plots are seen in Figure 3.



(a) $m + 1 = 20$ (b) $m + 1 = 40$ (c) $m + 1 = 80$ (d) $m + 1 = 160$

(e) $m + 1 = 320$ (f) $m + 1 = 640$ (g) $m + 1 = 1280$ (h) $m + 1 = 10240$

Figure 3: Plots of solution for varying numbers of mesh points