Math 151B Homework No. 4

Brandon Loptman

UID: 604105043

February 25, 2020

In [197]:
```
import numpy as np
import matplotlib.pyplot as plt
plt.style.use("ggplot")
```

In [198]:
```
def RK4_2D_System(f1,f2,t0,y0,h,N):
    """
    """

    T = np.array([t0 + n * h for n in range(N+1)])
    Y = np.zeros([N+1,2])

    Y[0,0] = y0[0]
    Y[0,1] = y0[1]

    for n in range(N):
        #print("n = ", n)
        #print("\n")

        k1  = h*f1(T[n],Y[n,0],Y[n,1])
        k1_ = h*f2(T[n],Y[n,0],Y[n,1])

        #print("k1: ", k1)
        #print("k1_: ", k1_)
        #print("\n")

        k2  = h*f1(T[n]+0.5*h,Y[n,0]+0.5*k1,Y[n,1]+0.5*k1_)
        k2_ = h*f2(T[n]+0.5*h,Y[n,0]+0.5*k1,Y[n,1]+0.5*k1_)

        #print("k2: ", k2)
        #print("k2_: ", k2_)
        #print("\n")


        k3  = h*f1(T[n]+0.5*h,Y[n,0]+0.5*k2,Y[n,1]+0.5*k2_)
        k3_ = h*f2(T[n]+0.5*h,Y[n,0]+0.5*k2,Y[n,1]+0.5*k2_)

        #print("k3: ", k3)
        #print("k3_: ", k3_)
        #print("\n")


        k4  = h*f1(T[n+1],Y[n,0]+k3,Y[n,1]+k3_)
        k4_ = h*f2(T[n+1],Y[n,0]+k3,Y[n,1]+k3_)

        #print("k4: ", k4)
        #print("k4_: ", k4_)
        #print("\n")

        Y[n+1,0] = Y[n,0] + (1/6)*(k1+ 2*k2 + 2*k3 + k4)
        Y[n+1,1] = Y[n,1] + (1/6)*(k1_+ 2*k2_ + 2*k3_ + k4_)

    return T,Y
```

In [199]:
```
def Linear_Shooting_Method(p,q,r,a,b,alpha,beta,N):
    """
    """

    h = (b-a)/N

    u0 = np.array([alpha,0])
    v0 = np.array([0,1])

    T = np.zeros(N+1)
    U = np.zeros([N+1,2])
    V = np.zeros([N+1,2])
    W = np.zeros([N+1,2])


    T,U = RK4_2D_System(f1,f2(p,q,r),a,u0,h,N)
    T,V = RK4_2D_System(f1,f2(p,q,r),a,v0,h,N)

    W[0,0] = alpha
    W[0,1] = (beta - U[N,0])/V[N,0]

    for n in range(1,N+1):
        W[n,0] = U[n,0] + W[0,1]*V[n,0]
        W[n,1] = U[n,1] + W[0,1]*V[n,1]

    return T,W
```

In [256]:
```
def f1(t,x1,x2):
    return x2

def f2(p, q, r):
    def innerFunc(t,x1,x2):
        return p(t)*x2 + q(t)*x1 + r(t)
    return innerFunc
```

11.1

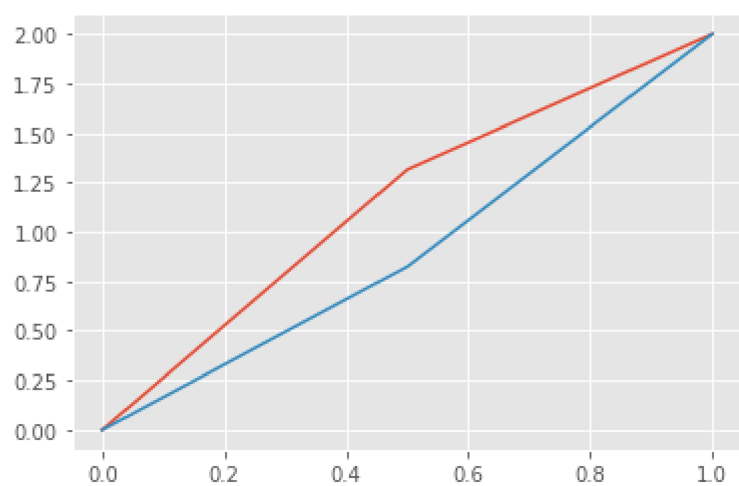
1.)

In [248]:
```
def p(t):
    return 0.0

def q(t):
    return 4.0

def r(t):
    return -4.0*t

def exact_y(t):
    return np.exp(2)*((np.exp(4)-1)**-1)*(np.exp(2*t)-np.exp(-2*t)) + t

a = 0.0
b = 1.0

alpha = 0.0
beta = 2.0
```

(a.)

In [202]:
```
N = 2

T,W = Linear_Shooting_Method(p,q,r,a,b,alpha,beta,N)
Y = exact_y(T)

print("T values: ", T)
print("Exact Y values: ",Y)
print("Approximate Y values: ", W[:,0])

plt.plot(T,W[:,0])
plt.plot(T,Y)
plt.show()
```

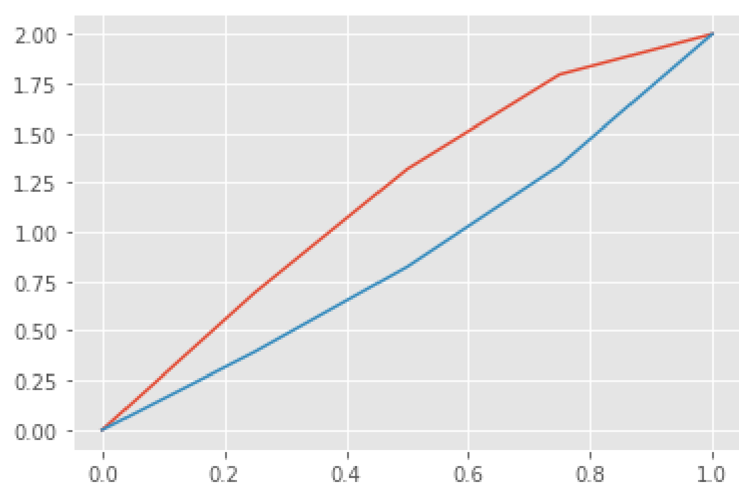
```
T values:  [0.  0.5 1. ]
Exact Y values:  [0.         0.82402714 2.        ]
Approximate Y values:  [0.         1.31597222 2.        ]
```



(b.)

In [203]:
```
N = 4

T,W = Linear_Shooting_Method(p,q,r,a,b,alpha,beta,N)
Y = exact_y(T)

print("T values: ", T)
print("Exact Y values: ",Y)
print("Approximate Y values: ", W[:,0])

plt.plot(T,W[:,0])
plt.plot(T,Y)
plt.show()
```

```
T values:  [0.   0.25 0.5  0.75 1.  ]
Exact Y values:  [0.         0.39367669 0.82402714 1.33708613 2.
]
Approximate Y values:  [0.         0.69260962 1.31875873 1.79508385 2.
]
```



In [189]:
```
def Nonlinear_Shooting_Method(a,b,alpha, beta, n, tol, M):
    """
    """

    w1 = np.zeros(n+1)
    w2 = np.zeros(n+1)
    X = np.zeros(n+1)
    h = (b-a)/n
    k = 1
    TK = (beta - alpha)/(b - a)

    while k <= M:

        w1[0] = alpha
        w2[0] = TK
        u1    = 0
        u2    = 1

        for i in range(1,n+1):
            x = a + (i-1)*h    #step 5

            t = x + 0.5*(h)

            k11 = h*w2[i-1]     #step 6

            k12 = h*f(x,w1[i-1],w2[i-1])
            k21 = h*(w2[i-1] + (1/2)*k12)
            k22 = h*f(t, w1[i-1] + (1/2)*k11, w2[i-1] + (1/2)*k12)
            k31 = h*(w2[i-1] + (1/2)*k22)
            k32 = h*f(t, w1[i-1] + (1/2)*k21, w2[i-1] + (1/2)*k22)
            t   = x + h
            k41 = h*(w2[i-1]+k32)
            k42 = h*f(t, w1[i-1] + k31, w2[i-1] + k32)
            w1[i] = w1[i-1] + (k11 + 2*k21 + 2*k31 + k41)/6
            w2[i] = w2[i-1] + (k12 + 2*k22 + 2*k32 + k42)/6
            kp11 = h*u2
            kp12 = h*(fy(x,w1[i-1],w2[i-1])*u1 + fyp(x,w1[i-1], w2[i-1])
*u2)
            t    = x + 0.5*(h)
            kp21 = h*(u2 + (1/2)*kp12)
            kp22 = h*((fy(t, w1[i-1],w2[i-1])*(u1 + (1/2)*kp11)) + fyp(x
+h/2, w1[i-1],w2[i-1])*(u2 +(1/2)*kp12))
            kp31 = h*(u2 + (1/2)*kp22)
            kp32 = h*((fy(t, w1[i-1],w2[i-1])*(u1 + (1/2)*kp21)) + fyp(x
+h/2, w1[i-1],w2[i-1])*(u2 +(1/2)*kp22))
            t    = x + h
            kp41 = h*(u2 + kp32)
            kp42 = h*(fy(t, w1[i-1], w2[i-1])*(u1+kp31) + fyp(x + h, w1[
i-1], w2[i-1])*(u2 + kp32))
            u1 = u1 + (1/6)*(kp11 + 2*kp21 + 2*kp31 + kp41)
            u2 = u2 + (1/6)*(kp12 + 2*kp22 + 2*kp32 + kp42)

        r = np.abs(w1[n] - beta)

        if r < tol:
            for i in range(0,n+1):
```

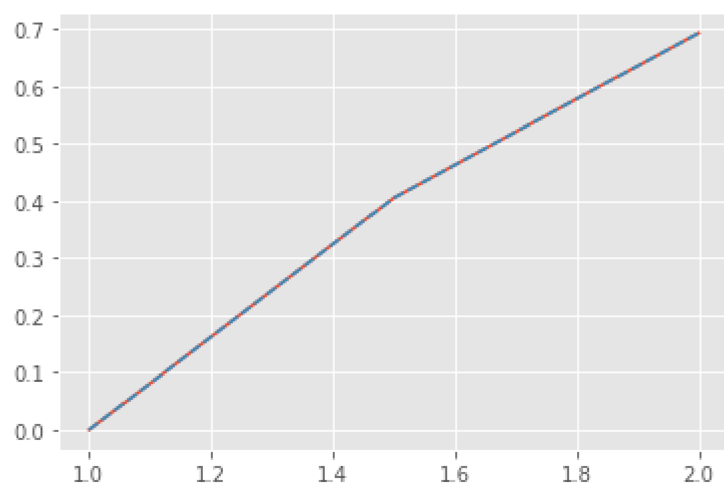
```
                X[i] = a + i*h

            return X, w1

        TK = TK -(w1[n]-beta)/u1

        k = k+1

    print("Maximum number of iterations exceeded")
    return X, w1
```

11.2

1.)

In [195]:
```
def f(x,y,yp):
    fx = -(yp**2)-y+np.log(x)
    return fx

def fy(xp,z,zp):
    fyy = -1
    return fyy

def fyp(xpp,zpp,zppp):
    fypp = -2*zppp
    return fypp

def y_exact(x):
    return np.log(x)

a = 1
b = 2
alpha = 0
beta = np.log(2)
N = 2
M = 100
tol = 1e-12


X,W = Nonlinear_Shooting_Method(a,b,alpha,beta,N,tol,M)
Y_exact = y_exact(X)

print("X values: ", X)
print("Exact Y values: " , Y_exact)
print("Approximate Y values: ", W)

plt.plot(X,W)
plt.plot(X,Y_exact,linestyle="--")
plt.show()
```

```
X values:  [1.  1.5 2. ]
Exact Y values:  [0.         0.40546511 0.69314718]
Approximate Y values:  [0.         0.40549954 0.69314718]
```



In [27]:
```
def F(f,x,n):
    fx = np.zeros([n,1])

    for i in range(n):
        fx[i] = f[i](x)

    return fx

def J(j,x,n):
    jx = np.zeros([n,n])

    for i in range(n):
        for k in range(n):
            jx[i,k] = j[i,k](x)

    return jx
```

In [219]:
```
def Newton_Method_Systems(n,x0,tol,N):
    k = 1
    x = x0
    while(k <= N):
        #print("iteration ", k)

        fx = F(f,x,n)
        jx = J(j,x,n)

        #print("J(x) = \n", jx)
        #print("F(x) = \n", fx)

        jx_inv = np.linalg.inv(jx)

        #print("F(x) = \n", fx)
        #print("J(x) = \n", jx)
        #print("J(x) Inverse = \n", jx_inv)


        y = -1.0*np.matmul(jx_inv,fx)
        y = y.reshape(n)

        x = x + y

        #print("y = ", y)
        #print("x = ", x)

        if(np.linalg.norm(y) < tol):
            print("The procedure was successful!")
            return x

        k = k + 1

    print("Max number of iterations surpassed. The procedure was unsucce
ssful!")
    return x
```

10.2

1.)

(a.)

In [220]:
```
def f1(x):
    return 4*(x[0]**2) - 20*x[0] + .25*(x[1]**2) + 8
def f2(x):
    return 0.5*x[0]*(x[1]**2) + 2*x[0] - 5*x[1] + 8
def j11(x):
    return 8*x[0] - 20
def j12(x):
    return 0.5*x[1]
def j21(x):
    return 0.5*x[1]**2 + 2
def j22(x):
    return x[0]*x[1] - 5

n = 2
tol = 10e-12
N = 2

f = np.array([f1,f2])
j = np.array([[j11,j12],[j21,j22]])
x0 = np.array([0,0])

x = Newton_Method_Systems(n,x0,tol,N)

print(x)
```

```
Max number of iterations surpassed. The procedure was unsuccessful!
[0.49589361 1.98342347]
```

(b.)

In [207]:
```
def f1(x):
    return np.sin(4*np.pi*x[0]*x[1]) - 2*x[1] - x[0]
def f2(x):
    return ((4*np.pi-1)/(4*np.pi))*(np.exp(2*x[0])-np.exp(1)) + 4*np.exp
(1)*(x[1]**2) - 2*np.exp(1)*x[0]
def j11(x):
    return 4*np.pi*x[1]*np.cos(4*np.pi*x[0]*x[1]) - 1
def j12(x):
    return 4*np.pi*x[0]*np.cos(4*np.pi*x[0]*x[1]) - 2
def j21(x):
    return ((4*np.pi-1)/(4*np.pi))*2*np.exp(2*x[0]) - 2*np.exp(1)
def j22(x):
    return 8*np.exp(1)*x[1]

n = 2
tol = 10e-12
N = 2

f = np.array([f1,f2])
j = np.array([[j11,j12],[j21,j22]])
x0 = np.array([0,0])

x = Newton_Method_Systems(n,x0,tol,N)

print(x)
```

```
Max number of iterations surpassed. The procedure was unsuccessful!
[-0.51316159 -0.01837622]
```

(c.)

In [208]:
```
def f1(x):
    return x[0]*(1-x[0]) + 4*x[1] - 12
def f2(x):
    return (x[0]-2)**2 + (2*x[1]-3)**2 - 25
def j11(x):
    return 1 - 2*x[0]
def j12(x):
    return 4
def j21(x):
    return 2*(x[0]-2)
def j22(x):
    return 4*(2*x[1]-3)

n = 2
tol = 10e-12
N = 2

f = np.array([f1,f2])
j = np.array([[j11,j12],[j21,j22]])
x0 = np.array([0,0])

x = Newton_Method_Systems(n,x0,tol,N)

print(x)
```

```
Max number of iterations surpassed. The procedure was unsuccessful!
[-23.94262597   7.60867966]
```

(d.)

The matrix $J(x^{(0)})$ is singular so Newton's method cannot be applied.

2.)

(a.)

In [237]:
```
def f1(x):
    return 3*x[0]-np.cos(x[2]*x[1])-0.5
def f2(x):
    return 4*(x[0]**2)-625*(x[1]**2)+2*x[1]-1
def f3(x):
    return np.exp(-x[0]*x[1])+20*x[2]+((10*np.pi-3)/3)
def j11(x):
    return 3
def j12(x):
    return x[2]*np.sin(x[1]*x[2])
def j13(x):
    return x[1]*np.sin(x[1]*x[2])
def j21(x):
    return 8*x[0]
def j22(x):
    return 1250*x[1] + 2
def j23(x):
    return 0
def j31(x):
    return -x[2]*np.exp(-x[1]*x[0])
def j32(x):
    return 0
def j33(x):
    return 20

n = 3
tol = 10e-6
N = 10

f = np.array([f1,f2,f3])
j = np.array([[j11,j12,j13],[j21,j22,j23],[j31,j32,j33]])
x0 = np.array([0,0,0])

x = Newton_Method_Systems(n,x0,tol,N)

print(x)
```

```
Max number of iterations surpassed. The procedure was unsuccessful!
[-8.30829761 19.03718281 -1.25349311]
```

(b.)

In [241]:
```
def f1(x):
    return (x[0]**2)+x[1]-37
def f2(x):
    return x[0]-(x[1]**2)-5
def f3(x):
    return x[0]+x[1]+x[2]-3
def j11(x):
    return 2*x[0]
def j12(x):
    return 1
def j13(x):
    return 0
def j21(x):
    return 1
def j22(x):
    return -4*x[1]
def j23(x):
    return 0
def j31(x):
    return 1
def j32(x):
    return 1
def j33(x):
    return 1

n = 3
tol = 10e-12
N = 2

f = np.array([f1,f2,f3])
j = np.array([[j11,j12,j13],[j21,j22,j23],[j31,j32,j33]])
x0 = np.array([0,0,0])

x = Newton_Method_Systems(n,x0,tol,N)

print(x)
```

```
Max number of iterations surpassed. The procedure was unsuccessful!
[  3.42606347  27.73936529 -28.16542876]
```

(c.)

In [246]:
```
def f1(x):
    return 15*x[0]+(x[1]**2)-4*x[2]-13
def f2(x):
    return (x[0]**2)+10*x[1]-x[2]-11
def f3(x):
    return (x[1]**3)-25*x[2]+22
def j11(x):
    return 15
def j12(x):
    return 2*x[1]
def j13(x):
    return -2
def j21(x):
    return 2*x[0]
def j22(x):
    return 10
def j23(x):
    return -1
def j31(x):
    return 0
def j32(x):
    return 3*(x[1]**2)
def j33(x):
    return -25

n = 3
tol = 10e-12
N = 2

f = np.array([f1,f2,f3])
j = np.array([[j11,j12,j13],[j21,j22,j23],[j31,j32,j33]])
x0 = np.array([0,0,0])

x = Newton_Method_Systems(n,x0,tol,N)

print(x)
```

```
Max number of iterations surpassed. The procedure was unsuccessful!
[1.02988426 1.08714296 0.92998579]
```

(d.)

The matrix $J(x^{(0)})$ is singular so Newton's method cannot be applied.

In [224]:
```
def f1(x):
    return 10*x[0]-2*(x[1]**2)+x[1]-2*x[2]-5
def f2(x):
    return 8*(x[1]**2)+4*(x[2]**3)-9
def f3(x):
    return 8*x[1]*x[2]+4
def j11(x):
    return 10
def j12(x):
    return -4*x[1]+1
def j13(x):
    return -2
def j21(x):
    return 0
def j22(x):
    return 16*x[1]
def j23(x):
    return 8*x[2]
def j31(x):
    return 0
def j32(x):
    return 8*x[2]
def j33(x):
    return 8*x[1]

n = 3
tol = 10e-12
N = 2

f = np.array([f1,f2,f3])
j = np.array([[j11,j12,j13],[j21,j22,j23],[j31,j32,j33]])
x0 = np.array([0,0,0])

x = Newton_Method_Systems(n,x0,tol,N)

print(x)
```

```
---------------------------------------------------------------------------
LinAlgError                               Traceback (most recent call last)
<ipython-input-224-963384e624d6> in <module>
     32 x0 = np.array([0,0,0])
     33 
---> 34 x = Newton_Method_Systems(n,x0,tol,N)
     35 
     36 print(x)

<ipython-input-219-82d910ef227b> in Newton_Method_Systems(n, x0, tol, N)
     11         #print("F(x) = \n", fx)
     12 
---> 13         jx_inv = np.linalg.inv(jx)
     14 
     15         #print("F(x) = \n", fx)

~/anaconda3/lib/python3.7/site-packages/numpy/linalg/linalg.py in inv(a)
    549     signature = 'D->D' if isComplexType(t) else 'd->d'
    550     extobj = get_linalg_error_extobj(_raise_linalgerror_singular)
--> 551     ainv = _umath_linalg.inv(a, signature=signature, extobj=extobj)
    552     return wrap(ainv.astype(result_t, copy=False))
    553 

~/anaconda3/lib/python3.7/site-packages/numpy/linalg/linalg.py in _raise_linalgerror_singular(err, flag)
     95 
     96 def _raise_linalgerror_singular(err, flag):
---> 97     raise LinAlgError("Singular matrix")
     98 
     99 def _raise_linalgerror_nonposdef(err, flag):

LinAlgError: Singular matrix
```

In [ ]: