Math 151B Final Exam Code

March 16, 2020

[71]:
```python
#importing all the necessary libraries
#these are all included in a standard installation of Anaconda

import numpy as np
import matplotlib.pyplot as plt
plt.style.use("ggplot")
```

0.1 3.)

(a.)

[199]:
```python
def EulerSystem3D(f,t0,y0,h,N):
    """
    """
    T = np.array([t0 + n * h for n in range(N + 1)])
    Y = np.zeros((N+1,3))

    Y[0] = y0

    for n in range(N):
        Y[n+1] =  Y[n] + h * f(T[n], Y[n])

    return T,Y
```

[219]:
```python
def f(t,y):
    return np.array([y[1],y[2],4*y[0]+4*y[1]-y[2]])

def Y_exact(t):
    return np.exp(-t) + np.exp(2*t) + np.exp(-2*t)

t0 = 0
y0 = np.array([3,-1,9])
h = .2
N = 10
```

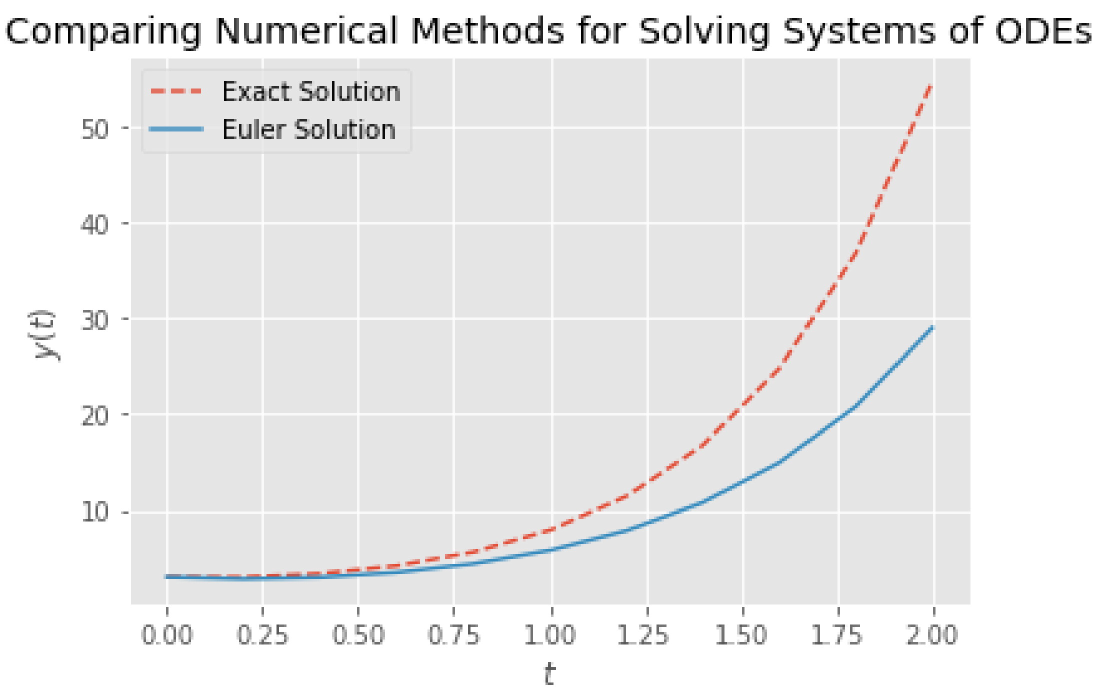
[230]:
```python
T,Y = EulerSystem3D(f,t0,y0,h,N)
print("T values: \n", T)
print("Approximate Y values: \n", Y[:,0])
```

```
T values:
 [0.  0.2 0.4 0.6 0.8 1.  1.2 1.4 1.6 1.8 2. ]
Approximate Y values:
 [ 3.          2.8         2.96        3.472       4.3808      5.78368
  7.838336   10.7790592  14.94245888 20.80534221 29.0388863 ]
```

[226]:
```python
plt.plot(T,y_exact(T),label="Exact Solution",linestyle="--")
plt.plot(T,Y[:,0],label="Euler Solution")

plt.title("Comparing Numerical Methods for Solving Systems of ODEs")
plt.ylabel("$y(t)$")
plt.xlabel("$t$")

plt.legend()
plt.show()
```



0.2 4.)

(a.)

[53]:
```python
A = np.array([[4,1,-1,0],[1,3,-1,0],[-1,-1,5,2],[0,0,2,4]])
#print(A)

evals = np.linalg.eigvals(A)
domEval = np.max(evals)

print("The dominant eigenvalue is ", domEval)
```

```
The dominant eigenvalue is  7.086130197651494
```

(b.)

[47]:
```python
def PowerMethod(A,x,n,tol=10e-4, N=25):
    """
    Implements the Power Method as described by the algorithm in the textbook.
    """
    k = 1

    xp = np.max(abs(x))
    x = x/xp

    eval_list = np.array([])

    while(k <= N):
        #print(k)
        y = A.dot(x)

        yp = np.max(np.abs(y))
        u = yp

        eval_list = np.append(eval_list,u)
        #print(eval_list)

        #print(u)

        if(yp == 0):
            print("A has eigenvalue 0, select a new vector x and restart.")
            return x,eval_list

        err = np.max(np.abs(x-(y/yp)))

        x = y/yp
        #print(x)

        if(err < tol):
            print("The procedure was successful!")
            return u,x,eval_list
```

```python
        k = k + 1

    print("The maximum number of iterations exceeded! The procedure was␣
↪unsuccessful.")
    return u,x,eval_list
```

[83]:
```python
n = 4
N = 25
tol = 10e-4
x0 = np.array([0,1,0,0])

u,v,evals_P = PowerMethod(A,x0,n,tol,N)

N1 = evals_P.size
N1_vals = np.arange(N1)

print("The dominant eigenvalue is ", u)
```

```
The procedure was successful!
The dominant eigenvalue is  7.087013746136077
```

(d.)

[84]:
```python
def SymmetricPowerMethod(A,x,n,tol=10e-4, N=25):
    """
    Implements the Symmetric Power Method as described by the algorithm in the␣
↪textbook.
    """
    k = 1

    x = x/np.linalg.norm(x)

    eval_list = np.array([])

    while(k <= N):
        #print(k)
        y = A.dot(x)

        u = x.dot(y)

        eval_list = np.append(eval_list,u)
        #print(eval_list)

        #print(u)

        y_norm = np.linalg.norm(y)

        if(y_norm == 0):
```

```python
            print("A has eigenvalue 0, select a new vector x and restart.")
            return x,eval_list

        err = np.linalg.norm(x - (y/y_norm))

        x = y/y_norm
        #print(x)

        if(err < tol):
            print("The procedure was successful!")
            return u,x,eval_list

        k = k + 1

    print("The maximum number of iterations exceeded! The procedure was␣
 ↪unsuccessful.")
    return u,x,eval_list
```

[82]:
```python
u,v,evals_S = SymmetricPowerMethod(A,x0,n,tol,N)

N2 = evals_S.size
N2_vals = np.arange(N2)

print("The dominant eigenvalue is ", u)
```

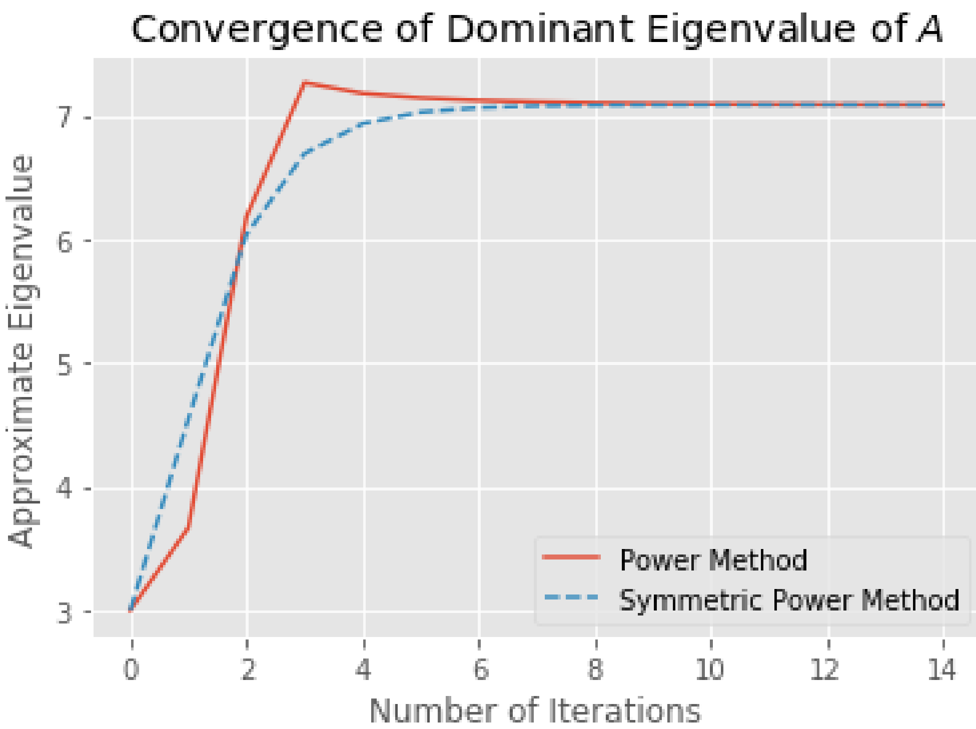
```
The procedure was successful!
The dominant eigenvalue is  7.086117416340071
```

(e.)

[85]:
```python
plt.plot(N1_vals,evals_P,label="Power Method")
plt.plot(N2_vals,evals_S,linestyle="--",label="Symmetric Power Method")

plt.title("Convergence of Dominant Eigenvalue of $A$")
plt.xlabel("Number of Iterations")
plt.ylabel("Approximate Eigenvalue")

plt.legend()
plt.show()
```



We can see from the plot that the Symmetric Power Method converges to the value of the domiant eigenvalue faster than the Power Method. Thus, the Symmetric Power Methods performs better.

0.3 5.)

[86]:
```python
def F(f,x,n):
    """
    Evaluates an n dimensional array of functions f at a point x.
    """
    fx = np.zeros([n,1])

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

    return fx

def J(j,x,n):
    """
    Evaluates an n x n matrix of functions j at a point x.
    """
    jx = np.zeros([n,n])
```

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

    return jx
```

(a.)

[87]:
```python
def G(f):
    """
    Computes the function g(x) as defined in the book.
    """
    return np.sum(f**2)

def Grad(J,F):
    """
    Computes the gradient of g(x) using the Jacobian and F.
    """
    n = F.shape[0]
    g = 2*np.matmul(np.transpose(J),F)
    return np.reshape(g,n)
```

[102]:
```python
def SteepestDescent(n,x0,tol,N):
    """
    Implements the Gradient Descent Algorithm according to the algorithm in the␣
↪book.
    """
    k = 1
    x = x0

    x_list = np.array([x])

    while(k <= N):
        #print(k)

        g1 = G(F(f,x,n))
        #print("g1 = ", g1)

        z = Grad(J(j,x,n),F(f,x,n))

        z0 = np.linalg.norm(z)

        if(z0 == 0):
            print("Zero gradient!")
            return x,x_list

        z = z/z0
```

```
    #print("z = ", z)

    a1 = 0
    a3 = 1

    g3 = G(F(f,x-a3*z,n))
    #print("g3 = ", g3)

    while(g3 >= g1):
        a3 = 0.5*a3
        g3 = G(F(f,x-a3*z,n))

        if(a3 < tol/2):
            print("No likely imporvement...")
            return x,x_list

    a2 = 0.5*a3

    g2 = G(F(f,x-a2*z,n))
    #print("g2 = ", g2)


    h1 = (g2-g1)/a2
    h2 = (g3-g2)/(a3-a2)
    h3 = (h2-h1)/a3

    #print("h1 = ", h1)
    #print("h2 = ", h2)
    #print("h3 = ", h3)


    a0 = 0.5*(a2-h1/h3)
    g0 = G(F(f,x-a0*z,n))

    #print("a0 = ", a0)
    #print("g0 = ", g0)


    if(g0 <= g3):
        a = a0
        g = g0
    else:
        a = a3
        g = g3

    x = x -a*z
    x_list = np.append(x_list,[x],axis=0)
```

```python
        if(np.abs(g-g1) < tol):
            print("The procedure was successful!")
            return x,x_list

        k = k+1

    print("Maximum number of iterations exceeded!")
    return x,x_list
```

[158]:
```python
def f1(x):
    return x[0]**3 + (x[0]**2)*x[1] - x[0]*x[2] + 6
def f2(x):
    return np.exp(x[0]) + np.exp(x[1]) - x[2]
def f3(x):
    return x[1]**2 - 2*x[0]*x[2] - 4

def j11(x):
    return 3*(x[0]**2) + 2*x[0]*x[1] - x[2]
def j12(x):
    return x[0]**2
def j13(x):
    return -1*x[0]
def j21(x):
    return np.exp(x[0])
def j22(x):
    return np.exp(x[1])
def j23(x):
    return -1
def j31(x):
    return -2*x[2]
def j32(x):
    return 2*x[1]
def j33(x):
    return -2*x[0]

f = np.array([f1,f2,f3])
j = np.array([[j11,j12,j13],[j21,j22,j23],[j13,j23,j33]])
x0 = np.array([1,1,1])

n = 3
tol = 10e-5
N = 100
```

[163]:
```python
x, xvals_SD = SteepestDescent(n,x0,tol,N)
print("The approximate solution to the system of equations is: \n","x = ", x)
```

```
No likely imporvement…
The approximate solution to the system of equations is:
 x =  [0.11822276 0.52917242 1.02811216]
```

(b.)

```
[160]: def Newton_Method_Systems(n,x0,tol,N):
    k = 1
    x = x0

    x_list = np.array([x])

    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*np.linalg.solve(jx,fx)
        y = y.reshape(n)

        x = x + y

        x_list = np.append(x_list,[x],axis=0)

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

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

        k = k + 1

    print("Max number of iterations surpassed. The procedure was unsuccessful!")
    return x,x_list
```

[162]:

```python
x,xvals_NM = Newton_Method_Systems(n,x0,tol,N)
print("The approximate solution to the system of equations is: \n","x = ", x)
```

```
Max number of iterations surpassed. The procedure was unsuccessful!
The approximate solution to the system of equations is:
 x =  [-2.80897387  1.74159019  2.3547992 ]
```

(c.)

[170]:

```python
def BroydenMethod(n, x0, tol = 10e-6, N = 100):
    """
    Implements Broyden's Method according to the algorithm in the book.
    """
    x = x0.reshape((n,1))
    x_list = np.array([x.flatten()])

    A0 = J(j,x,n)
    v = F(f,x,n)

    A = np.linalg.inv(A0)

    s = -1*np.matmul(A,v)
    x = x + s
    k = 2

    while(k <= N):
        w = v
        v = F(f,x,n)
        y = v - w

        z = -1*np.matmul(A,y)

        p = -1*np.matmul(np.transpose(s),z)

        ut = np.matmul(np.transpose(s),A)

        A = A + (1/p)*np.matmul((s+z),ut)

        s = -1*np.matmul(A,v)
        x = x + s

        x_list = np.append(x_list,[x.flatten()],axis=0)

        if(np.linalg.norm(s) < tol):
            print("The procedure was successful!")
            return x.flatten(),x_list

        k = k + 1
```

```
    print("Maximum number of iterations exceeded!")
    return x.flatten(),x_list
```

[172]:
```
x,xvals_BM = BroydenMethod(n,x0,tol,N)
print("The approximate solution to the system of equations is: \n","x = ", x)
```

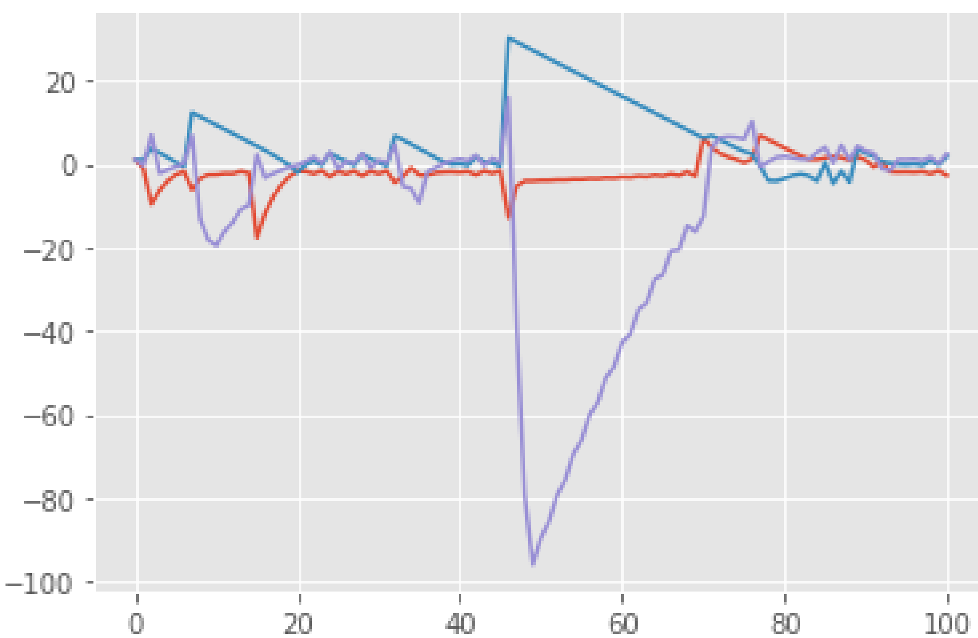
```
The procedure was successful!
The approximate solution to the system of equations is:
 x =  [-1.4560397  -1.66423221  0.42249382]
```

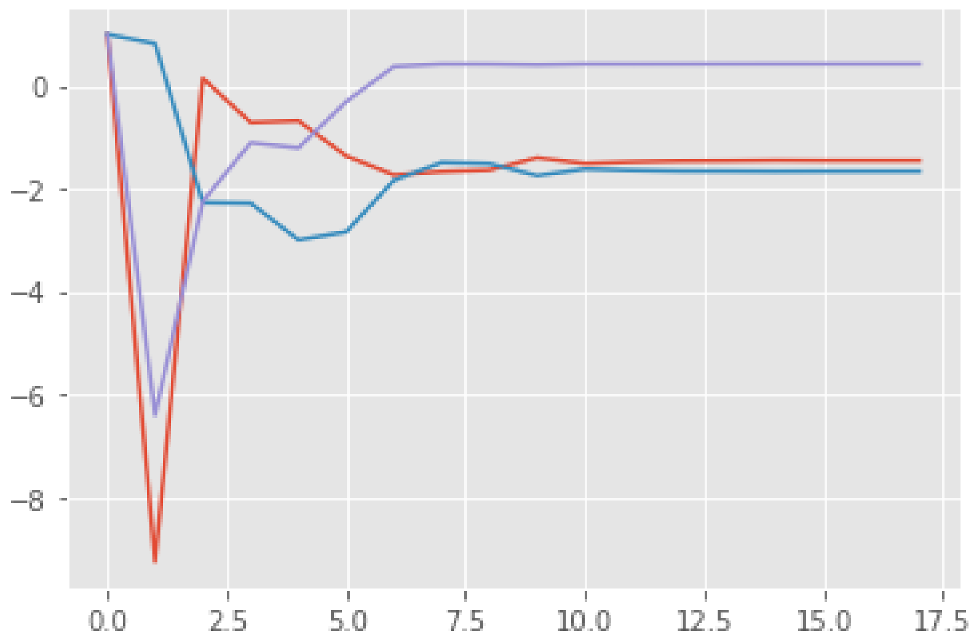
(d.)

[188]:
```
N_NM = np.shape(xvals_NM)[0]
N_BM = np.shape(xvals_BM)[0]

NM_vals = np.arange(N_NM)
BM_vals = np.arange(N_BM)

plt.plot(NM_vals,xvals_NM)
plt.show()
plt.plot(BM_vals,xvals_BM)
plt.show()
```




0
−2
−4
−6
−8
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

[ ]: