

Applied Mathematics TW324 Assignment 03

Bhekimpilo Ndhlela (18998712)

22 March 2018

Question 1

a.)

Python Source Code:

```
#!/usr/bin/python
def question_a(debug=True):
    global J
    J = [bessel_function(i) for i in xrange(0, 4)]

    if debug is True:
        print "\nDebug_Mode: ON\tQuestion_1(a.)"
        print "i", "\t", "Jv(1)"
        for i in xrange(0, len(J)):
            print i, "\t", "{:.10f}".format(J[i])

def bessel_function(v, x=1, j=0):
    b = lambda k : pow(1, k) * pow(float(x)/2, v + (2 * k)) \
        / (sm.factorial(k) * sm.factorial(v + k))
    for k in xrange(0, 4):
        j = j + b(k)
    return j
```

v	0.0	1.0	2.0	3.0
$J_v(1)$	0.7651909722	0.4400499132	0.1149034288	0.0195633500

b.)

Python Source Code:

```
def question_b(debug=True):
    x = linspace(0,3, num=100) # equally spaced points on interval [0, 3]
    x = [a for a in x if a != 0. if a != 1. if a != 2. if a != 3.]
    # the interpolating function from Barycentric Interpolation
    num = lambda v : (J[0]/v) ((3*J[1])/(v 1.))+((3*J[2])/(v 2.)) \
                    (J[3]/(v 3.))
    den = lambda v : (1./v) (3./(v 1.))+ (3./(v 2.)) (1./(v 3.))
    P = [num(i) / den(i) for i in x]
    global J
    J = [bessel_function(i) for i in x]

    # plot p(v) and Jv(1) on the same system
    func1, = plt.plot(x, J, label="Jv(1)", linestyle=' ')
    func2, = plt.plot(x, P, label="P(v)", linestyle=' ')
    plt.title('Jv(1) and P(v)')
    plt.ylabel('Jv(1) and P(v)')
    plt.xlabel('v')
    first_legend = plt.legend(handles=[func1], loc=1)
    ax = plt.gca().add_artist(first_legend)
    plt.legend(handles=[func2], loc=4)
    plt.show()
    # plot the error function Jv(1) - p(v)
    error = [jv - pv for jv, pv in zip(J, P)]
    plt.plot(x, error, 'r')
    plt.title('Error Function')
    plt.ylabel('Error: Jv(1) - P(v)')
    plt.xlabel('v')
    plt.show()

    if debug is False:
        print "\nDebug Mode: ON \t Question 1 (b.)"
        print "i \t x \t P(x) \t Jx(1) \t err"
        for i in xrange(len(x)):
            print i, "\t", "{:.10f}".format(x[i]), "\t", \
                  "{:.10f}".format(P[i]), "\t", \
                  "{:.10f}".format(J[i]), "\t", \
                  "{:.10f}".format(error[i])

    return error
```

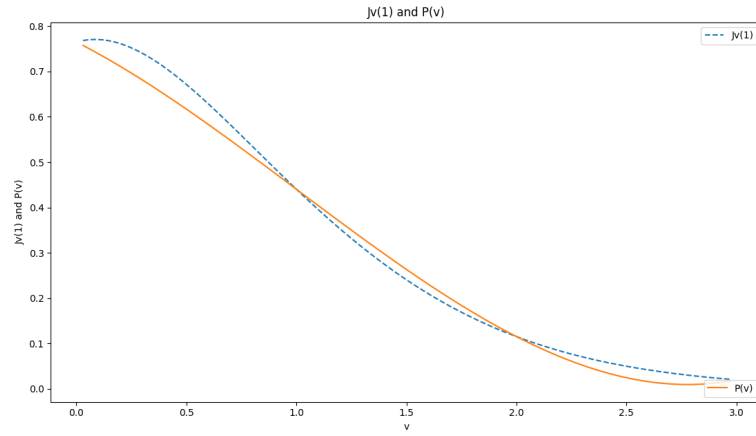


Figure 1: These are the $J_v(1)$ and $P(v)$ curves/ graphs

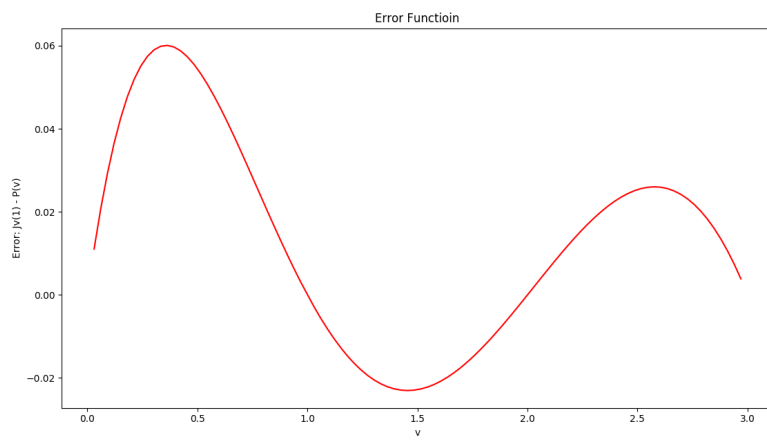


Figure 2: This is the error function: $J_v(1) - P(v)$

c.)

Python Source Code:

```
def question_c(error , M4=3.0, h=1.0, n=4.0, debug=True):
    est_error = (1. / (4. * n)) * (h**n) * M4
    max_error = sorted(error)[ 1] # get max error from question 1 b.)
    #compare errors is est_error >= max_error
    is_bound_true = est_error >= max_error

    if debug is True:
        print "\nDebug_Mode: ON\tQuestion_1_c)"
        print "Estimated_Error\t:", est_error
        print "Maximum_Error [Question_1_c.)] \t:", max_error
        print "est_error >= max_error? \t", is_bound_true
```

Estimated Error	Maximum Error [Question 1 b.)]
0.187500000000000	0.0600749992108

Theoretic Estimated Max Error \geq Max Error [Question 1 b.)] ?

By taking $h = 1.0$, I proved that:

$$C_n h^n M_n \geq \max_{x_1 \leq x \leq x_4} |f(x) - P_{n-1}(x)|$$

d.)

Python Source Code:

```
def question_d(debug=True):
    # coeff of : pi'(x) = 2x^3 - 9x^2 + 11x - 3 = 0
    coeff = [2, -9, 11, -3]
    zeros = roots(coeff)

    pi_x = lambda x : x**4 - 6*(x**3) + 11*(x**2) - 6*x
    maxi = [pi_x(x) for x in zeros]
    if debug is True:
        print "\nDebug_Mode: ON\n\tQuestion_1(d.)"
        for i, (zero, max_min) in enumerate(zip(zeros, maxi)):
            print i, '\t', "{:.10f}".format(zero), '\t', \
                  "{:.10f}".format(max_min), '\t', \
                  "{:.10f}".format(fabs(max_min))
```

$$\pi(x) = x(x-1)(x-2)(x-3)$$

$$\pi(x) = x^4 - 6x^3 + 11x^2 - 6x$$

$$\pi'(x) = 4x^3 - 18x^2 + 22x - 6$$

$$\text{let: } \pi'(x) = 0$$

$$0 = 4x^3 - 18x^2 + 22x - 6$$

Therefore:

$$x_1, \quad x_2, \quad x_3 = 2.6180339887, \quad 1.5000000000, \quad 0.3819660113 \quad \text{respectively}$$

Hence:

$$\pi(x_1) = \pi(2.6180339887) = -1.0000000000 \quad |\pi(2.6180339887)| = 1.0000000000$$

$$\pi(x_2) = \pi(1.5000000000) = 0.5625000000 \quad |\pi(1.5000000000)| = 0.5625000000$$

$$\pi(x_3) = \pi(0.3819660113) = -1.0000000000 \quad |\pi(0.3819660113)| = 1.0000000000$$

$$\max(1.0000000000; 0.5625000000; 1.0000000000) = 1.0000000000$$

$$\text{Hence: } \max_{x_0 \leq x \leq x_3} |\pi(x)| = 1.0000$$

Question 3

a.)

Python Source Code for)

```
def question_a(N=4, debug=True):
    x = zeros(N)
    x = [cos(((2 * k - 1) * pi)/(2 * N)) for k in xrange(1, N + 1)]
    # defining the chebyshev Vandermonde matrix where V = 4x4 Matrix
    V = C.chebvander(x, N - 1).T
    if debug is True:
        print "DEBUG_MODE: ON [ Question 3 (a.) ]:"
        print "vector x====", x
        print "\nchebyshev Vandermonde matrix where V:====\n", V
    return x, V

x = [0.9238795325112867  0.38268343236508984  -0.3826834323650897  -0.9238795325112867]
```

let: **V = Chebyshev Vandermonde Matrix**

$$v = \begin{pmatrix} 1.0 & 1.0 & 1.0 & 1.0 \\ 0.92387953 & 0.38268343 & -0.38268343 & -0.92387953 \\ 0.70710678 & -0.70710678 & -0.70710678 & 0.70710678 \\ 0.38268343 & -0.92387953 & 0.92387953 & -0.38268343 \end{pmatrix}$$

b.)

Python Source Code:

```
def question_b(x, V, debug=True):
    # finding the e^xi for the x elements
    exp_x = [exp(i) for i in x]
    V_inv = inv(V)
    c = matmul(V_inv, exp_x)
    if debug is True:
        print "\nDEBUG_MODE: ON [ Question_3(b.) ]:"
        print "vector_e^x=", exp_x
        print "\n(chebyshev_Vandermonde_matrix)^ I_where_V:\n", V_inv
        print "\nVc=exp(x)=>c=inv(V)exp(c)<=>inv(V)V=I:\n", c
```

$e^x = [2.5190441714069842 \quad 1.4662138007571095 \quad 0.682028773350537 \quad 0.39697596864348]$

$$Vc = e^x$$

$$V^{-1}Vc = V^{-1}e^x$$

$$c = V^{-1}e^x$$

$$c = \begin{pmatrix} 0.25 & 0.46193977 & 0.35355339 & 0.19134172 \\ 0.25 & 0.19134172 & -0.35355339 & -0.46193977 \\ 0.25 & -0.19134172 & -0.35355339 & 0.46193977 \\ 0.25 & -0.46193977 & 0.35355339 & -0.19134172 \end{pmatrix} \begin{pmatrix} 2.5190441714069842 \\ 1.4662138007571095 \\ 0.682028773350537 \\ 0.39697596864348 \end{pmatrix}$$

$$c = ([1.62415515 \quad 0.48579634 \quad 0.29145858 \quad 0.1176341])$$

The c represents
.....

c.)

Python Source Code:

```
def question_c(b=3, a=0, N=0.0, debug=True):
    approx = lambda N : exp(N) * ((b - a)**N / (fact(N) * 2**(2*N - 1)))
    while 10**10 <= approx(N):
        N = N + 1.0
        print N
    if debug is True:
        print "\nDEBUG_MODE: ON [ Question_3(c.) ]:"
        print "Number of Chebyshev Points N=", int(N)
        print "Approximated Value, @N=", approx(N)
    return N
```

The number of Chebyshev Points necessary to approximate the function $\exp(x)$ to an accuracy of 10^{-10} on the interval $[0, 3]$ is :

$$n = 19$$

Approximated Value at, $n = 19$ is $1.24077928729e - 11$

d.)

Python Source Code:

```
def question_d(N, debug=True):
    chebyshev_points = zeros(int(N))
    chebyshev_points = [3./2. + 3./2.*cos(((2 * k - 1) * pi)/(2 * N)) \
                        for k in xrange(1, int(N) + 1)]
    if debug is True:
        print "\nDEBUG_MODE: ON [Question_d.]"
        print "The Chebyshev Point where n=", int(N)
        print "i", "\t", "Chebyshev Points"
        for i, value in enumerate(chebyshev_points):
            print i, "\t", value
    return chebyshev_points
```

i	Chebyshev Points
1	2.99487673951
2	2.95410039891
3	2.87365998998
4	2.75574971739
5	2.60358586601
6	2.42131906903
7	2.21392108956
8	1.98704920381
9	1.74689188542
10	1.50000000000
11	1.25310811458
12	1.01295079619
13	0.786078910444
14	0.578680930965
15	0.39641413399
16	0.244250282606
17	0.126340010017
18	0.045899601091
19	0.0051232604

e.)

Python Source Code:

```
def question_e(chebyshev_points, debug=True):
    exp_xk = [exp(vk) for vk in chebyshev_points]
    #use polyfit
    fit = polyfit(chebyshev_points, exp_xk, len(exp_xk) - 1)
    err_vals = [e - f for e, f in zip(exp_xk, fit)]
    warnings.simplefilter('ignore', RankWarning) #ignore warnings
    # plot the error function
    plt.plot(linspace(0,3,num=19), exp_xk, label="exp(xk)")
    plt.plot(linspace(0,3,num=19), err_vals, label="err_vals")
    plt.plot(linspace(0,3,num=19), fit, label="fit")
    plt.legend(bbox_to_anchor=(1.0, 1), loc=0, borderaxespad=0.)
    plt.show()
    if debug is True:
        print "\nDEBUG_MODE: ON [Question 3.e.)"
        print "k", "\t", "chebyshev_points(k)", "\t", "exp(k)\t\t\t", \
              "polyfit_points(k)"
        for k, (xk, e) in enumerate(zip(chebyshev_points, exp_xk)):
            print k, "\t", "{:.16f}".format(xk), "\t", "{:.16f}".format(e), \
                  "\t", "{:.16f}".format(fit[k])
```

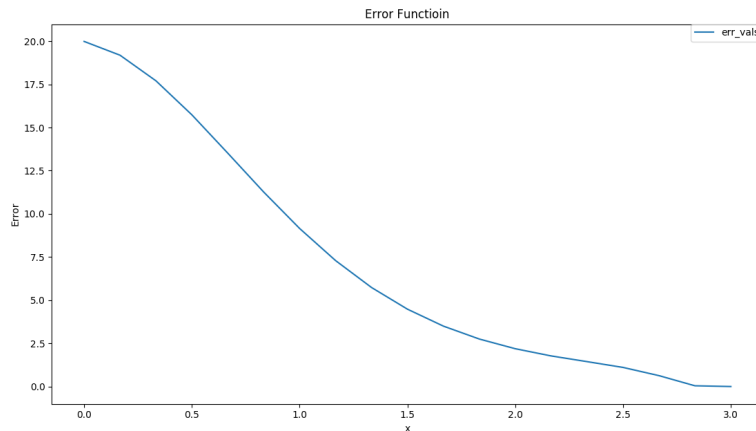


Figure 3: Error of e^x

k	chebyshev'points(k)	exp(k)	polyfit'points(k)
1	2.9948767395100049	19.9828966364183991	-0.00000000000000168
2	2.9541003989089956	19.1844565966013114	0.00000000000004260
3	2.8736599899825861	17.7016877825937016	-0.00000000000047462
4	2.7557497173937930	15.7328316599325380	0.00000000000335313
5	2.6035858660096975	13.5121038604438937	-0.0000000001397079
6	2.4213190690345021	11.2607031675513909	0.0000000006593969
7	2.2139210895556101	9.1515301018746555	0.0000000008709762
8	1.9870492038070253	7.2939789307318259	0.0000000272904685
9	1.7468918854211006	5.7367444785709703	0.0000002724300402
10	1.5000000000000000	4.4816890703380645	0.0000027591227060
11	1.2531081145788994	3.5012082197865295	0.0000247987718511
12	1.0129507961929751	2.7537146890514030	0.0001984144914214
13	0.7860789104443897	2.1947736279721375	0.0013888880233439
14	0.5786809309654983	1.7836840758813131	0.0083333336421386
15	0.3964141339903027	1.4864847939769930	0.0416666665889665
16	0.2442502826062074	1.2766638172542299	0.166666666793699
17	0.1263400100174137	1.1346679011556191	0.499999999988355
18	0.0458996010910044	1.0469692911054875	1.000000000000446
19	0.0051232604899951	1.0051364068301394	1.000000000000009