PHYS2160 Introductory Computational Physics
2021/22 Solutions to Exercise 5

1. ```
   # relaxation.py
   # This program uses the relaxation method to find a root of f(x) =
   # x^3 - 2x - 2 to an accuracy of 10^(-12) starting from the initial
   # value x0 = 1.
   # Last update on 12 Apr 2021 by F K Chow

   def g(x):
       """ Compute the function g(x) = (2x + 2)^(1/3) """
       return (2*x + 2)**(1/3)

   def dgdx(x):
       """ Compute the derivative of g(x) = (2x + 2)^(1/3) """
       return (2/3)*(2*x + 2)**(-2/3)

   # Use the relaxation method to find the root of f(x)
   x = 1
   tol = 1e-12
   error = 1
   xi = []
   errxi = []
   while error > tol:
       xo = x
       x = g(xo)
       error = abs((xo - x)/(1 - 1.0/dgdx(xo)))
       xi.append(x)
       errxi.append(error)

   # Output the results of computation by this method
   print("{:>3s} {:>16s} {:>17s}".format("i", "xi", "|Error(xi)|"))
   for i in range(len(xi)):
       print("{:3d} {:16.12f} {:17.13f}".format(i+1, xi[i], errxi[i]))
   ```

   Below is the output of this program:

   ```
     i               xi       |Error(xi)|
     1   1.587401051968   0.2113133458996
     2   1.729675293391   0.0407935522028
     3   1.760814725540   0.0085297090207
     4   1.767485065145   0.0018096392365
     5   1.768907380001   0.0003850807121
     6   1.769210364154   0.0000819949122
     7   1.769274892996   0.0000174614620
     8   1.769288635586   0.0000037186625
     9   1.769291562293   0.0000007919460
    10   1.769292185581   0.0000001686572
    11   1.769292318320   0.0000000359182
    12   1.769292346589   0.0000000076493
    13   1.769292352610   0.0000000016290
    14   1.769292353892   0.0000000003469
    15   1.769292354165   0.0000000000739
   ```
   ```

```
 16   1.769292354223   0.0000000000157
 17   1.769292354235   0.0000000000034
 18   1.769292354238   0.0000000000007
```

2.
```
# relaxationvar.py
# This program uses the relaxation method to find a solution of x =
# f(x) = exp(exp(-x)*cos(x)) to an accuracy of 10^(-12) starting from
# the initial value x0 = 1 without using the formula of f'(x).
# Written on 25 Mar 2020 by F K Chow

from math import cos, exp

def f(x):
    """ Compute the function f(x) = exp(exp(-x)*cos(x)) """
    return exp(exp(-x)*cos(x))

# Use the relaxation method to find the solution of x = f(x) without
# using the formula of f'(x)
xo = 1
x = f(xo)
tol = 1e-12
error = 1
xi = [x]
errxi = [error]
while error > tol:
    xoo = xo
    xo = x
    x = f(xo)
    error = abs((xo - x)**2/(2*xo - xoo - x))
    xi.append(x)
    errxi.append(error)

# Output the results of computation by this method
print("{:>3s} {:>16s} {:>17s}".format("i", "xi", "|Error(xi)|"))
print("{:3d} {:16.12f} {:>17s}".format(1, xi[0], "n/a"))
for i in range(1, len(xi)):
    print("{:3d} {:16.12f} {:17.13f}".format(i+1, xi[i], errxi[i]))
```

Below is the output of this program:

```
  i               xi       |Error(xi)|
  1   1.219896611335               n/a
  2   1.106822998315   0.0383987546818
  3   1.159453227253   0.0167162146959
  4   1.133616658189   0.0085071433777
  5   1.145995891927   0.0040100015362
  6   1.139992777064   0.0019604343736
  7   1.142887242444   0.0009415964352
  8   1.141487749057   0.0004561249530
  9   1.142163505490   0.0002200442335
 10   1.141836999138   0.0001063657147
 11   1.141994708290   0.0000513659236
```

12	1.141918520046	0.0000248170722
13	1.141955323387	0.0000119874939
14	1.141937544610	0.0000057909977
15	1.141946132948	0.0000027974067
16	1.141941984172	0.0000013513531
17	1.141943988317	0.0000006527949
18	1.141943020175	0.0000003153459
19	1.141943487855	0.0000001523339
20	1.141943261933	0.0000000735879
21	1.141943371069	0.0000000355481
22	1.141943318348	0.0000000171722
23	1.141943343816	0.0000000082954
24	1.141943331513	0.0000000040072
25	1.141943337456	0.0000000019358
26	1.141943334586	0.0000000009351
27	1.141943335972	0.0000000004517
28	1.141943335302	0.0000000002182
29	1.141943335626	0.0000000001054
30	1.141943335470	0.0000000000509
31	1.141943335545	0.0000000000246
32	1.141943335509	0.0000000000119
33	1.141943335526	0.0000000000057
34	1.141943335518	0.0000000000028
35	1.141943335522	0.0000000000013
36	1.141943335520	0.0000000000006

3.
```
# bisection.py
# This program uses the bisection method to find a root of f(x) =
# x*tan(x) - sqrt(100 - x^2) on the interval 6.5 <= x <= 7.5 to an
# accuracy of 10^(-12).
# Written on 25 Mar 2020 by F K Chow

from math import sqrt, tan

def f(x):
    """ Compute the function f(x) = x*tan(x) - sqrt(100 - x^2) """
    return x*tan(x) - sqrt(100 - x**2)

a, b = 6.5, 7.5
if f(a)*f(b) > 0:
    print("Bisection method fails for this case!")
else:
    # Use the bisection method to find the root of f(x)
    x1, x2 = a, b
    tol = 1e-12
    error = 1
    xm = a
    xi = []
    delxi = []

    while error > tol:
        xmo = xm
        xm = 0.5*(x1 + x2)
```

```
    if f(xm)*f(x1) > 0:
        x1 = xm
    else:
        x2 = xm
    error = abs(x1 - x2)
    xi.append(xm)
    delxi.append(abs(xm - xmo))
xmo = xm
xm = 0.5*(x1 + x2)
xi.append(xm)
delxi.append(abs(xm - xmo))

# Output the results of computation by this method
print("{:>3s} {:>16s} {:>17s}".format("i", "x_i", "|x_i - x_(i-1)|"))
print("{:3d} {:16.12f} {:>17s}".format(1, xi[0], "n/a"))
for i in range(1, len(xi)):
    print("{:3d} {:16.12f} {:17.13f}".format(i+1, xi[i], delxi[i]))
```

Below is the output of this program:

```
  i              x_i   |x_i - x_(i-1)|
  1   7.000000000000               n/a
  2   7.250000000000   0.2500000000000
  3   7.125000000000   0.1250000000000
  4   7.062500000000   0.0625000000000
  5   7.093750000000   0.0312500000000
  6   7.078125000000   0.0156250000000
  7   7.070312500000   0.0078125000000
  8   7.066406250000   0.0039062500000
  9   7.068359375000   0.0019531250000
 10   7.069335937500   0.0009765625000
 11   7.068847656250   0.0004882812500
 12   7.069091796875   0.0002441406250
 13   7.068969726562   0.0001220703125
 14   7.068908691406   0.0000610351562
 15   7.068878173828   0.0000305175781
 16   7.068893432617   0.0000152587891
 17   7.068885803223   0.0000076293945
 18   7.068889617920   0.0000038146973
 19   7.068891525269   0.0000019073486
 20   7.068890571594   0.0000009536743
 21   7.068891048431   0.0000004768372
 22   7.068891286850   0.0000002384186
 23   7.068891167641   0.0000001192093
 24   7.068891227245   0.0000000596046
 25   7.068891257048   0.0000000298023
 26   7.068891242146   0.0000000149012
 27   7.068891234696   0.0000000074506
 28   7.068891238421   0.0000000037253
 29   7.068891236559   0.0000000018626
 30   7.068891237490   0.0000000009313
 31   7.068891237024   0.0000000004657
 32   7.068891237257   0.0000000002328
 33   7.068891237373   0.0000000001164
 34   7.068891237315   0.0000000000582
 35   7.068891237344   0.0000000000291
 36   7.068891237330   0.0000000000146
 37   7.068891237337   0.0000000000073
```

38	7.068891237341	0.0000000000036
39	7.068891237343	0.0000000000018
40	7.068891237343	0.0000000000009
41	7.068891237343	0.0000000000005

4.
```
# newtons.py
# This program uses the Newton's method to find a root of f(x) =
# 2*sin(3x) - exp(x) to an accuracy of 10^(-12) starting from the
# initial value x0 = 0.
# Written on 28 Mar 2022 by F K Chow

from math import cos, exp, sin

def f(x):
    """ Compute the function f(x) = 2*sin(3x) - exp(x) """
    return 2*sin(3*x) - exp(x)

def dfdx(x):
    """ Compute the derivative of f(x) = 2*sin(3x) - exp(x) """
    return 6*cos(3*x) - exp(x)

# Use the Newton's method to find the root of f(x)
x = 0
delx = f(x)/dfdx(x)
tol = 1e-12
error = 1
xi = []
errxi = []
while error > tol:  # Error of x_i = x_(i+1) - x_i!!
    x -= delx
    xi.append(x)
    delx = f(x)/dfdx(x)
    error = abs(delx)
    errxi.append(error)

# Output the results of computation by this method
print("{:>3s} {:>16s} {:>17s}".format("i", "xi", "|Error(xi)|"))
for i in range(len(xi)):
    print("{:3d} {:16.12f} {:17.13f}".format(i+1, xi[i], errxi[i]))
```

Here is the output of this program:

i	xi	|Error(xi)|
1	0.200000000000	0.0246924198386
2	0.224692419839	0.0010432224941
3	0.225735642333	0.0000019876530
4	0.225737629986	0.0000000000072
5	0.225737629993	0.0000000000000

5.
```
# secant.py
# This program uses the Secant method to find a root of f(x) =
```

```
# exp(x^2)*ln(x^2) - x to an accuracy of 10^(-12) starting from the
# initial values x0 = 1 and x1 = 2.
# Written on 28 Mar 2022 by F K Chow

from math import exp, log

def f(x):
    """ Compute the function f(x) = exp(x^2)*ln(x^2) - x """
    return exp(x**2)*log(x**2) - x

# Use the Secant method to find the root of f(x)
xo, x = 1, 2
delx = f(x)*(x - xo)/(f(x) - f(xo))
tol = 1e-12
error = 1
xi = []
errxi = []
while error > tol:  # Error of x_i = x_(i+1) - x_i!!
    xo = x
    x -= delx
    xi.append(x)
    delx = f(x)*(x - xo)/(f(x) - f(xo))
    error = abs(delx)
    errxi.append(error)

# Output the results of computation by this method
print("{:>3s} {:>16s} {:>17s}".format("i", "xi", "|Error(xi)|"))
for i in range(len(xi)):
    print("{:3d} {:16.12f} {:17.13f}".format(i+2, xi[i], errxi[i]))
```

Below is the output of this program:

i	xi	|Error(xi)|
2	1.013388833168	0.0124153225717
3	1.025804155739	0.1843924756801
4	1.210196631419	0.0615026051210
5	1.148694026298	0.0124128967890
6	1.161106923087	0.0013357867157
7	1.162442709803	0.0000364585821
8	1.162406251221	0.0000000956887
9	1.162406346910	0.0000000000070
10	1.162406346917	0.0000000000000

6.
```
# relaxationsimeq.py
# This program uses the relaxation method to solve the simultaneous
# equations x^2 - 2x + y^4 - 2y^2 + y = 0 and x^2 + x + 2y^3 - 2y^2
# - 1.5y - 0.05 = 0 starting from the initial values x = 0 and y = 1.
# It outputs each successive estimate of x and y until they both have
# absolute change less than 10^(-10).
# Written on 27 Jan 2021 by F K Chow
```

```
# Use the relaxation method to find the root of the simultaneous
# equations
x = 0
y = 1
tol = 1e-10
delx = dely = 1
xi = []
yi = []
while delx > tol or dely > tol:
    xo = x
    yo = y
    x = (xo**2 + yo**4 - 2*yo**2 + yo)/2.0
    y = (xo**2 + xo + 2*yo**3 - 2*yo**2 - 0.05)/1.5
    delx = abs(x - xo)
    dely = abs(y - yo)
    xi.append(x)
    yi.append(y)

# Output the results of computation by this method
print("{:>3s} {:>14s} {:>14s}".format("i", "xi", "yi"))
for i in range(len(xi)):
    print("{:3d} {:14.10f} {:14.10f}".format(i+1, xi[i], yi[i]))
```

Below is the output of this program:

```
  i             xi             yi
  1   0.0000000000  -0.0333333333
  2  -0.0177771605  -0.0348641975
  3  -0.0184888586  -0.0466512755
  4  -0.0253286921  -0.0484685080
  5  -0.0262599196  -0.0530755107
  6  -0.0290060057  -0.0543355896
  7  -0.0296951187  -0.0562601373
  8  -0.0308493624  -0.0569999162
  9  -0.0312678290  -0.0578440297
 10  -0.0317735104  -0.0582460665
 11  -0.0320051046  -0.0586295831
 12  -0.0322341483  -0.0588391359
 13  -0.0323560989  -0.0590177354
 14  -0.0324624363  -0.0591243310
 15  -0.0325248372  -0.0592089065
 16  -0.0325750704  -0.0592623320
 17  -0.0326064553  -0.0593028298
 18  -0.0326304660  -0.0593293607
 19  -0.0326460846  -0.0593488935
 20  -0.0326576512  -0.0593619922
 21  -0.0326653724  -0.0593714576
 22  -0.0326709728  -0.0593779006
 23  -0.0326747738  -0.0593825013
 24  -0.0326774944  -0.0593856631
 25  -0.0326793606  -0.0593879036
 26  -0.0326806850  -0.0593894529
 27  -0.0326815997  -0.0593905453
 28  -0.0326822454  -0.0593913037
 29  -0.0326826932  -0.0593918368
 30  -0.0326830082  -0.0593922078
```

```
31  -0.0326832273  -0.0593924681
32  -0.0326833811  -0.0593926495
33  -0.0326834883  -0.0593927766
34  -0.0326835634  -0.0593928653
35  -0.0326836158  -0.0593929274
36  -0.0326836525  -0.0593929708
37  -0.0326836781  -0.0593930011
38  -0.0326836960  -0.0593930223
39  -0.0326837085  -0.0593930371
40  -0.0326837173  -0.0593930475
41  -0.0326837234  -0.0593930547
42  -0.0326837277  -0.0593930598
43  -0.0326837307  -0.0593930633
44  -0.0326837327  -0.0593930658
45  -0.0326837342  -0.0593930675
46  -0.0326837352  -0.0593930687
47  -0.0326837359  -0.0593930696
48  -0.0326837364  -0.0593930702
49  -0.0326837368  -0.0593930706
50  -0.0326837370  -0.0593930709
51  -0.0326837372  -0.0593930711
52  -0.0326837373  -0.0593930712
53  -0.0326837374  -0.0593930713
```

7.
```python
# circuit.py
# This program uses the Newton's method to solve the simultaneous
# equations (V1 - Vp)/R1 + V1/R2 + I0(exp((V1- V2)/VT) - 1) = 0 and
# and (V2 - Vp)/R3 + V2/R4 - I0(exp((V1 - V2)/VT) - 1) = 0 to an
# accuracy of 10^(-8)V starting from the initial values V10 = 2V and
# V20 = 1V. Note that V1 and V2 are the voltages on the two sides of
# a diode in a circuit.
# Written on 22 Mar 2022 by F K Chow

import numpy as np

# Set up the parameters for the simultaneous equations in SI units
R1 = 1000
R2 = 4000
R3 = 3000
R4 = 2000
Vp = 5
VT = 0.05
I0 = 3e-9

def f(x):
    """ Compute the functions f1(V1, V2) = (V1 - Vp)/R1 + V1/R2 +
        I0(exp((V1- V2)/VT) - 1) and f2(V1, V2) = (V2 - Vp)/R3 + V2/R4
        - I0(exp((V1 - V2)/VT) - 1) """
    f = np.zeros(2)
    f[0] = (x[0] - Vp)/R1 + x[0]/R2 + I0*(np.exp((x[0] - x[1])/VT) - 1)
    f[1] = (x[1] - Vp)/R3 + x[1]/R4 - I0*(np.exp((x[0] - x[1])/VT) - 1)
    return f
```

```
def dfdx(x):
    """ Compute the partial derivatives of f1(V1, V2) = (V1 - Vp)/R1 +
        V1/R2 + I0(exp((V1- V2)/VT) - 1) and f2(V1, V2) = (V2 - Vp)/R3
        + V2/R4 - I0(exp((V1 - V2)/VT) - 1) """
    dfdx = np.zeros((2, 2))
    dfdx[0,0] = 1/R1 + 1/R2 + (I0/VT)*np.exp((x[0] - x[1])/VT)
    dfdx[0,1] = -(I0/VT)*np.exp((x[0] - x[1])/VT)
    dfdx[1,0] = -(I0/VT)*np.exp((x[0] - x[1])/VT))
    dfdx[1,1] = 1/R3 + 1/R4 + (I0/VT)*np.exp((x[0] - x[1])/VT)
    return dfdx

# Use the Newton's method to find the root of the simultaneous
# equations
x = np.array([2.0, 1.0])
delx = np.linalg.solve(dfdx(x), f(x))
tol = 1e-8
error = 1
xi = []
delxi = []
while error > tol:
    x -= delx
    xi.append(x.copy())
    delx = np.linalg.solve(dfdx(x), f(x))
    error = np.linalg.norm(delx)
    delxi.append(error)

# Output the results of computation by this method
print("{:>3s} {:>12s} {:>12s} {:>14s}".format("i", "V1i", "V2i",
                                               "||Delta(Xi)||"))
for i in range(len(xi)):
    print("{:3d} {:12.8f} {:12.8f} {:14.9f}".format(i+1, xi[i][0],
                                                     xi[i][1], delxi[i]))
```

Below is the output of this program:

```
  i          V1i          V2i  ||Delta(Xi)||
  1   3.58000721   2.62998918    0.036018493
  2   3.56002775   2.65995838    0.035950429
  3   3.54008604   2.68987094    0.035758413
  4   3.52025084   2.71962374    0.035221920
  5   3.50071323   2.74893015    0.033758318
  6   3.48198749   2.77701877    0.030008261
  7   3.46534190   2.80198715    0.021840628
  8   3.45322690   2.82015965    0.009747339
  9   3.44782005   2.82826993    0.001528015
 10   3.44697246   2.82954131    0.000032146
 11   3.44695463   2.82956806    0.000000014
 12   3.44695462   2.82956807    0.000000000
```