Scientific Computing assignment 7

Alfred Ajay Aureate R

December 21, 2018

Problem 1.a) The following is the code for the python module **pFit.py** with functions **pFit** for fitting data to a linear equation and **VanderMonde** function computes the Vander Monde matrix for a set of points.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Dec 19 19:06:25 2018

@author: alfred_mac
"""

import numpy as np
import numpy.linalg as la

def pFit(X,f):
    V = VanderMonde(X)
    p = la.solve(V,f)
    return p

def VanderMonde(X):
    d = len(X)
    VM = np.zeros([d,d])
    for i in range(d):
        for j in range(d):
            VM[i][j] = X[i]**j

    return VM
```

The following module evaluates the function value at the interpolation points using Horner's rule for the polynomisl calculation:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
```

```
Created on Wed Dec 19 19:21:28 2018

@author: alfred_mac
"""

import numpy as np
import numpy.linalg as la

def pEval(x,p):
    d = len(p)
    f = np.zeros(d)#0*np.arange(d)
    for i in range(d):
        f[i] = Horner(x[i],p)

    return f

def Horner(x,p):
    d = len(p)
    S = 1 + (p[d-1]/p[d-2])*x
    for i in range(d-2):
        S = 1 + (p[d-2-i]/p[d-3-i])*x*S
    S = p[0]*S
    return S
```

The following code is used to test both the **pFit.py** and **pEval.py** modules for 4 different functions with 100 data points or points where interpolation has to happen:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Dec 19 19:55:30 2018

@author: alfred_mac
"""

import numpy as np
import matplotlib.pyplot as plt
import numpy.linalg as la
import numpy.random as rand
import pFit
import pEval

d = 100
x = 2*rand.random(d)-1
```

```
# Test 1 - Function is 1/(1+x)
f = 1/(1+x)
p = pFit.pFit(x,f)
fp = pEval.pEval(x,p)

# Test 2 - Function is e^x
f = np.exp(x)
p = pFit.pFit(x,f)
fp = pEval.pEval(x,p)

# Test 3 - Function is tan(x^3)
f = np.tan(x**3)
p = pFit.pFit(x,f)
fp = pEval.pEval(x,p)

# Test 4 - Function is 3(x^2)/(1+sin(x))
f = 3*x*x/(1+np.sin(x))
p = pFit.pFit(x,f)
fp = pEval.pEval(x,p)

plt.plot(x,f,'b.')
plt.plot(x,fp,'r+')
plt.title('Test 4 - $f(x)=3x^{2}/(1+sin(x))$ and its $f_{p}(x)$ vs $x$')
plt.legend(['$f(x)$','$f_{p}(x)$'])
plt.xlabel('x')
plt.ylabel('$f(x)$ and $f_{p}(x)$')
plt.show()
```

Problem 1.b) The following code calculates the condition number of the Vander Monde matrix for different values of d:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Dec 19 20:48:28 2018

@author: alfred_mac
"""



import numpy as np
import matplotlib.pyplot as plt
import numpy.linalg as la
import numpy.random as rand
import pFit
import pEval
```

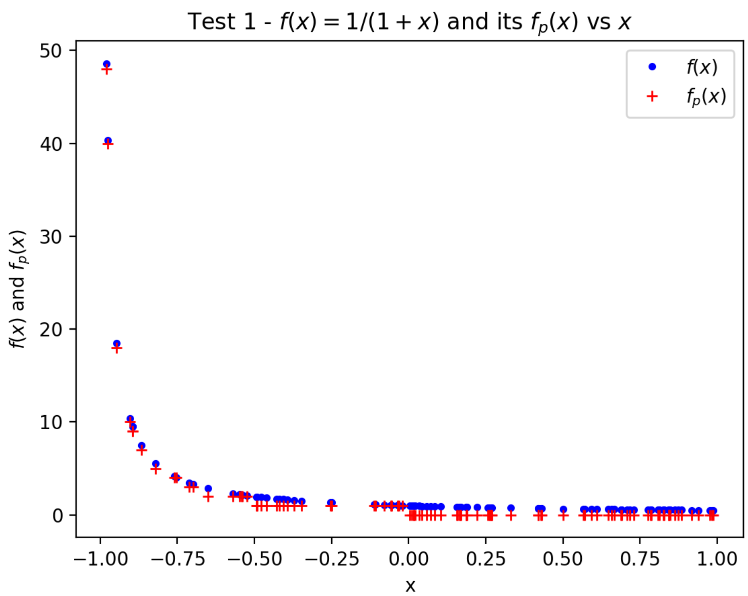

Figure 1: Test 1 - $f(x) = 1/(1+x)$ and its $f_p(x)$ vs x

```
D = 30
CN = 0*np.arange(D)

for i in range(D):
    x = 2*rand.random(i+1)-1
    V = pFit.VanderMonde(x)
    CN[i] = la.cond(V)

plt.semilogy(np.arange(D)+1,CN)
plt.title('Condition number vs d')
plt.xlabel('d')
plt.ylabel('Condition number')
plt.show()
```

We see that it's better to plot the relation on the semilogy plot which is log scale on the y-axis and linear scale on the x-axis. This suggests that condition number exponentially increases with d.

Problem 1.c) The following code fits the function $F(x) = e^{-0.5x^2}$ using polynomial interpolation for points in the interval $[-2, 2]$:

```
#!/usr/bin/env python3
```

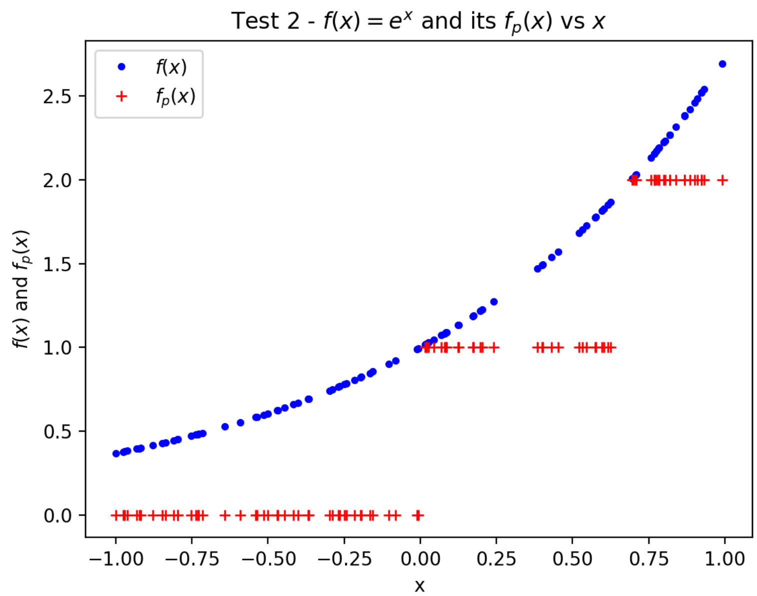

Figure 2: Test 2 - $f(x) = e^x$ and its $f_p(x)$ vs x

```
# -*- coding: utf-8 -*-
"""
Created on Wed Dec 19 21:04:15 2018

@author: alfred_mac
"""

import numpy as np
import matplotlib.pyplot as plt
import numpy.linalg as la
import numpy.random as rand
import pFit
import pEval

d = 100 # Note: Error increases as d increases too much
x = 4*rand.random(d)-2

# Function is e^(-0.5x^2)
f = np.exp(-0.5*x*x)
p = pFit.pFit(x,f)
fp = pEval.pEval(x,p)
```

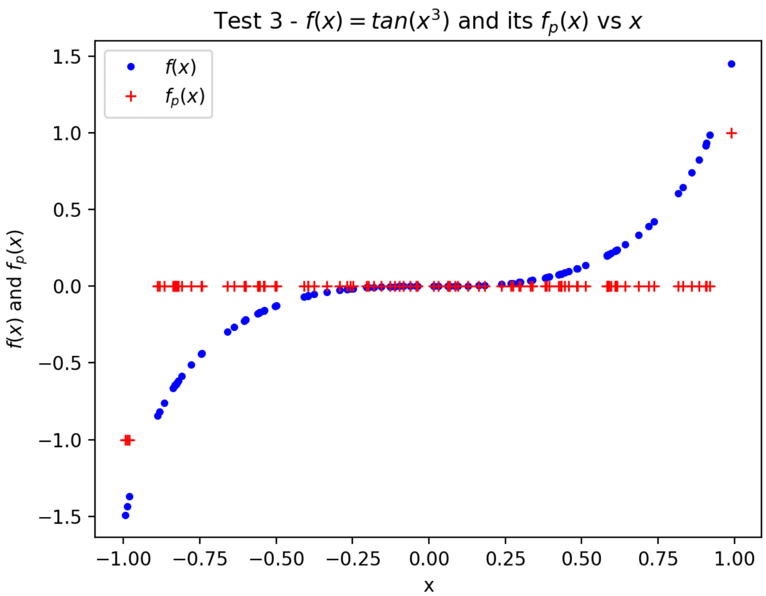

Figure 3: Test 3 - $f(x) = tan(x^3)$ and its $f_p(x)$ vs x

```
plt.plot(x,f,'b .')
plt.plot(x,fp,'r +')
plt.title('$f(x)=e^{-0.5x^{2}}$ and its $f_{p}(x)$ vs $x$')
plt.legend(['$f(x)$','$f_{p}(x)$'])
plt.xlabel('x')
plt.ylabel('$f(x)$ and $f_{p}(x)$')
plt.show()
```

Also, error seems to increase as d increases.

Problem 3.a) The following is the code for the python module **rFit.py** with functions **rFit** for fitting data to a linear equation and **matricize** function computes the matrix M to be used in the linear equation.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Dec 21 14:43:09 2018

@author: alfred_mac
"""
```

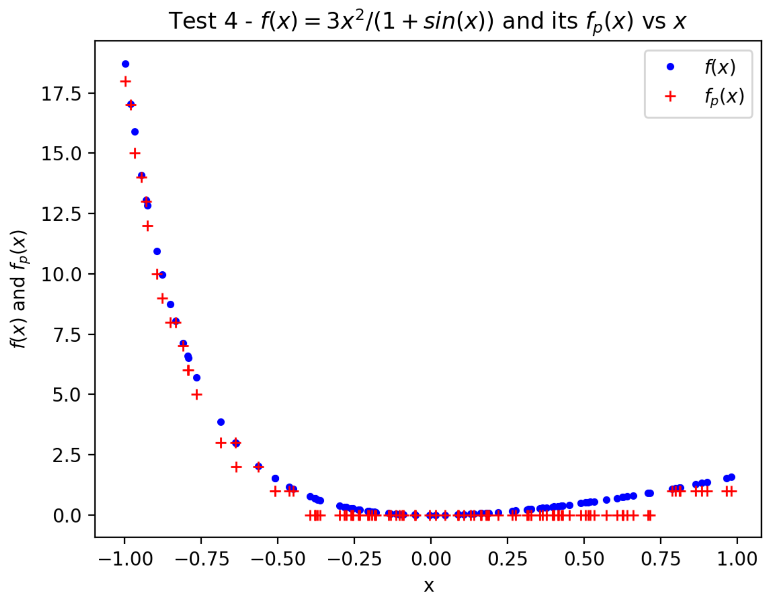

Figure 4: Test 4 - $f(x) = 3x^2/(1 + sin(x))$ and its $f_p(x)$ vs x

```
import numpy as np
import numpy.linalg as la

def rFit(X,f,L):
    V = matricize(X,L)
    p = la.solve(V,f)
    return p

def matricize(X,L):
    d = len(X)
    M = np.zeros([d,d])
    for i in range(d):
        for j in range(d):
            M[i][j] = np.exp(-0.5*(X[i]-X[j])*(X[i]-X[j])/(L*L))

    return M
```

The following module evaluates the function value at the interpolation points using radial functions as the basis:

```
#!/usr/bin/env python3
```

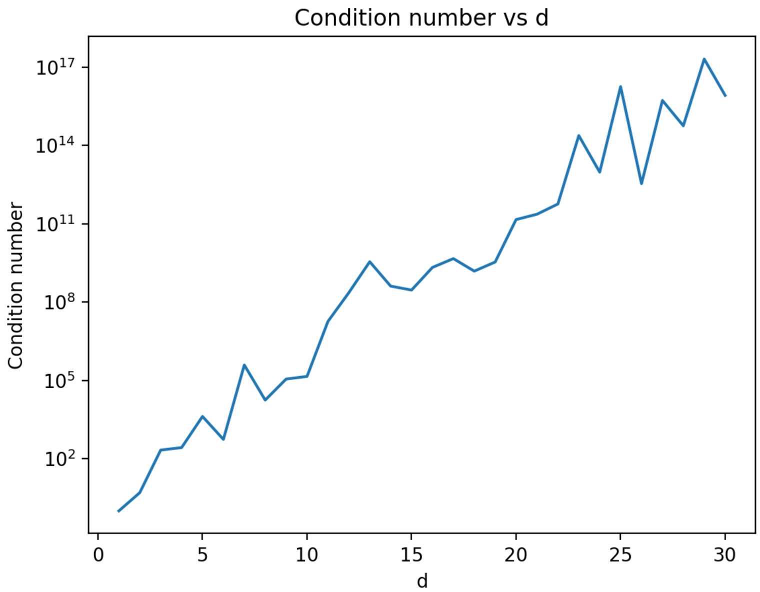

Figure 5: Condition number of the matrix M used in the linear equation vs d

```
# -*- coding: utf-8 -*-
"""
Created on Fri Dec 21 16:18:13 2018

@author: alfred_mac
"""

import numpy as np
import numpy.linalg as la
import rFit

def rEval(X,W,L):
    d = len(W)
    f = np.matmul(rFit.matricize(X,L),W)

    return f
```

The following code for **rTest.py** is used to test both the **rFit.py** and **rEval.py** modules for 4 different functions with 100 data points or points where interpolation has to happen:

```
#!/usr/bin/env python3
```

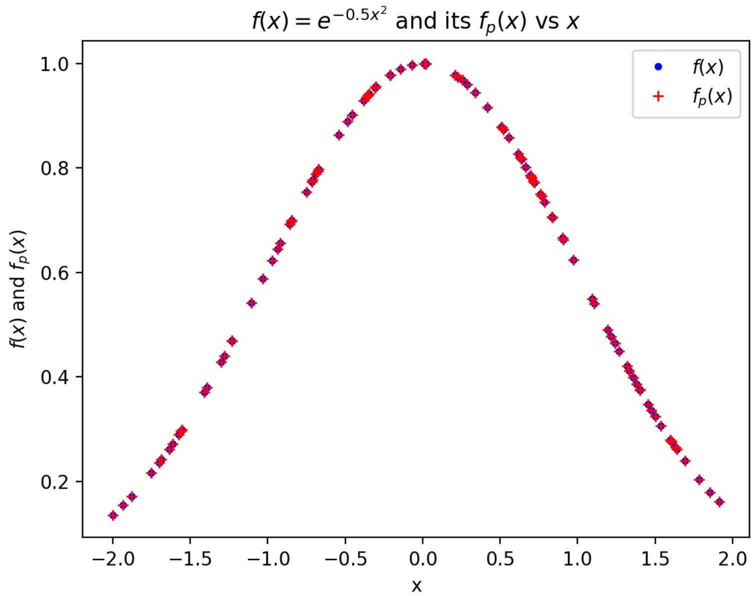

Figure 6: Interpolation between $F(x) = e^{-0.5x^2}$

```
# -*- coding: utf-8 -*-
"""
Created on Fri Dec 21 16:21:33 2018

@author: alfred_mac
"""


import numpy as np
import matplotlib.pyplot as plt
import numpy.linalg as la
import numpy.random as rand
import rFit
import rEval

d = 100
x = 2*rand.random(d)-1
L = 2
'''
# Test 1 - Function is e^(-0.5*x^2)
f = np.exp(-0.5*x*x)
p = rFit.rFit(x,f,L)
```

```
fp = rEval.rEval(x,p,L)

# Test 2 - Function is e^x
f = np.exp(x)
p = rFit.rFit(x,f,L)
fp = rEval.rEval(x,p,L)
'''
# Test 3 - Function is tan(x^3)
f = np.tan(x**3)
p = rFit.rFit(x,f,L)
fp = rEval.rEval(x,p,L)

# Test 4 - Function is 3(x^2)/(1+sin(x))
f = 3*x*x/(1+np.sin(x))
p = rFit.rFit(x,f,L)
fp = rEval.rEval(x,p,L)

plt.plot(x,f,'b.')
plt.plot(x,fp,'r+')
plt.title('Test 4 - $f(x)=3x^{2}/(1+sin(x))$ and its $f_{p}(x)$ vs $x$')
plt.legend(['$f(x)$','$f_{p}(x)$'])
plt.xlabel('x')
plt.ylabel('$f(x)$ and $f_{p}(x)$')
plt.show()
```

Problem 3.b) The following code calculates the condition number of the matrix M used in the linear equation for different values of d and L:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Dec 21 17:00:23 2018

@author: alfred_mac
"""


import numpy as np
import matplotlib.pyplot as plt
import numpy.linalg as la
import numpy.random as rand
import rFit

D = [10,100,1000]
L = [0.002,0.02,0.2]
CN = np.zeros([len(D),len(L)])
```

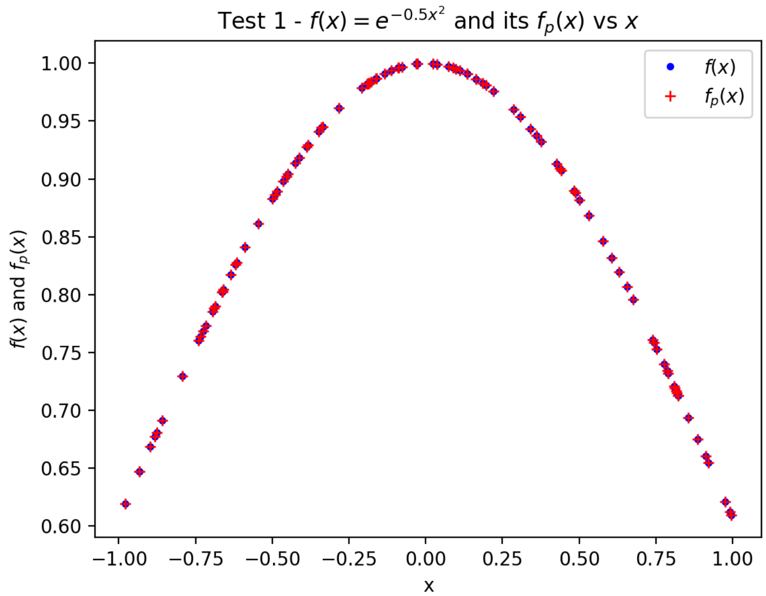

Figure 7: Test 1 - $f(x) = 1/(1 + x)$ and its $f_p(x)$ vs x

```
for i in range(len(D)):
    for j in range(len(L)):
        X = 2*rand.random(D[i])-1
        CN[i][j] = la.cond(rFit.matricize(X,L[j]))



plt.plot(CN)
plt.title('Condition number vs d and L')
plt.legend(['$L=0.002$','$L=0.02$','$L=0.2$','$L=2$'])
plt.xlabel('d')
plt.ylabel('Condition number')
plt.show()
```

Problem 3.c) The following code fits the function $F(x) = e^{-0.5x^2}$ using radial basis function interpolation for points in the interval $[-1, 1]$:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Dec 21 17:27:37 2018
```

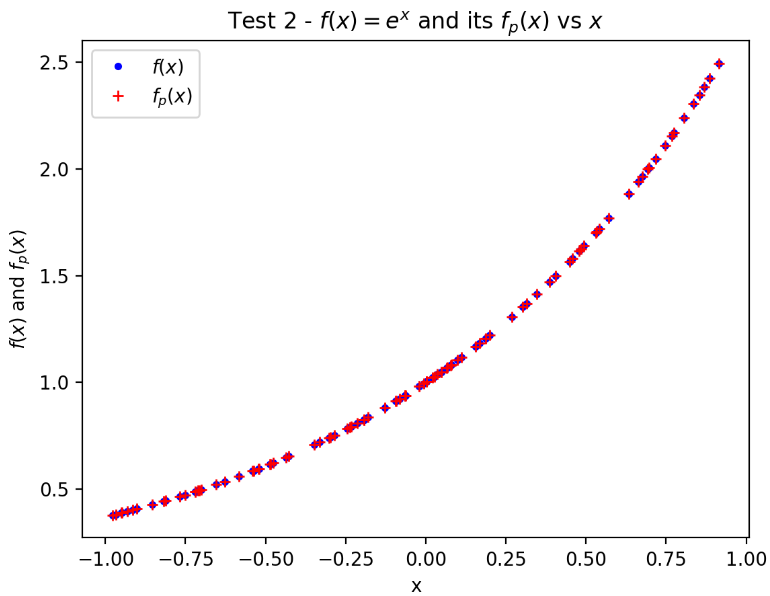

Figure 8: Test 2 - $f(x) = e^x$ and its $f_p(x)$ vs x

```
@author: alfred_mac
"""

import numpy as np
import matplotlib.pyplot as plt
import numpy.linalg as la
import numpy.random as rand
import rFit
import rEval

D = [101,1001]
L = [0.002,0.02,0.2,2,20]
Err = np.zeros([len(D),len(L)])

for i in range(len(D)):
    for j in range(len(L)):
        x = 2*rand.random(D[i])-1
        f = np.exp(-0.5*x*x)
        p = rFit.rFit(x,f,L[j])
        fp = rEval.rEval(x,p,L[j])
        Err[i][j] = max(abs(f-fp))
```

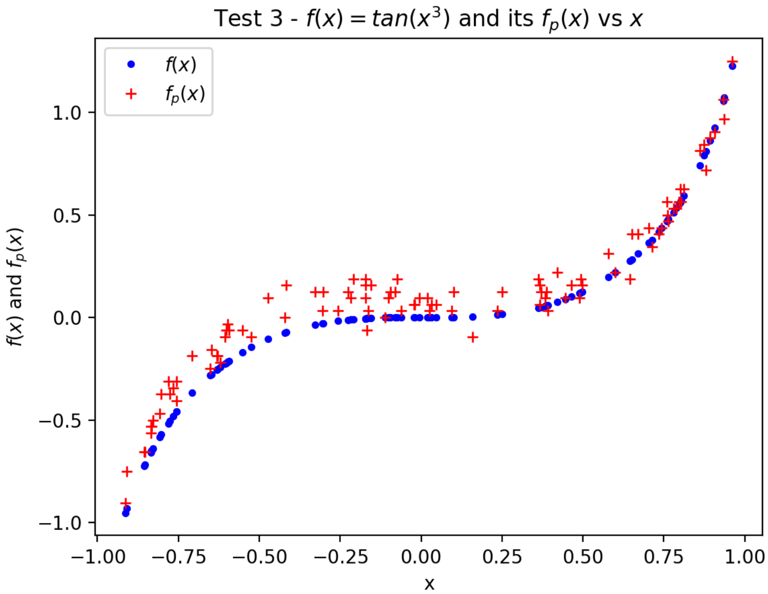

Figure 9: Test 3 - $f(x) = tan(x^3)$ and its $f_p(x)$ vs x

```
i,j = np.unravel_index(Err.argmin(), Err.shape)
x = 2*rand.random(D[i])-1
f = np.exp(-0.5*x*x)
p = rFit.rFit(x,f,L[j])
fp = rEval.rEval(x,p,L[j])

print("Optimal d is: ",D[i]-1)
print("Optimal L is: ",L[j])

plt.plot(x,f,'b .')
plt.plot(x,fp,'r +')
plt.title('$f(x)=e^{-0.5x^{2}}$ and its $f_{p}(x)$ vs $x$, with optimized d and
plt.legend(['$f(x)$','$f_{p}(x)$'])
plt.xlabel('x')
plt.ylabel('$f(x)$ and $f_{p}(x)$')
plt.show()
```

Since, lower d and lower L gives better accuracy, here d is chosen to be 100 and L to be 0.002.

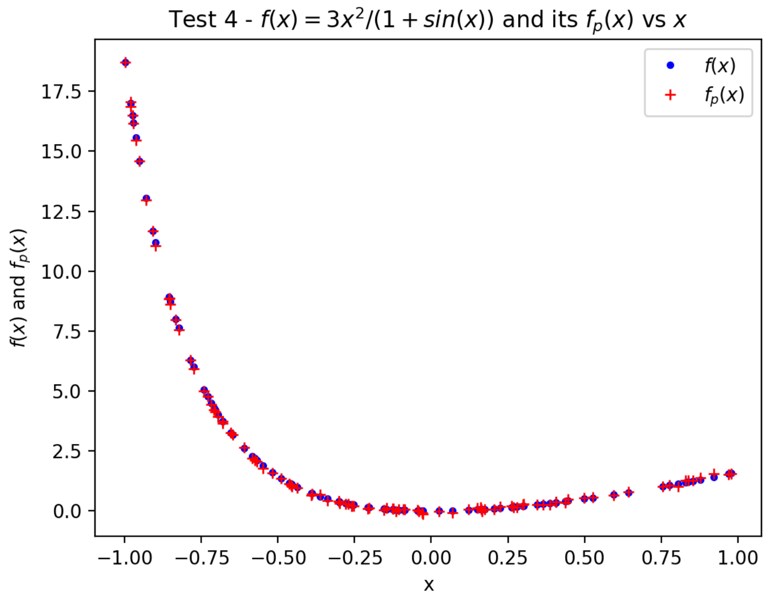

Figure 10: Test 4 - $f(x) = 3x^2/(1 + sin(x))$ and its $f_p(x)$ vs x

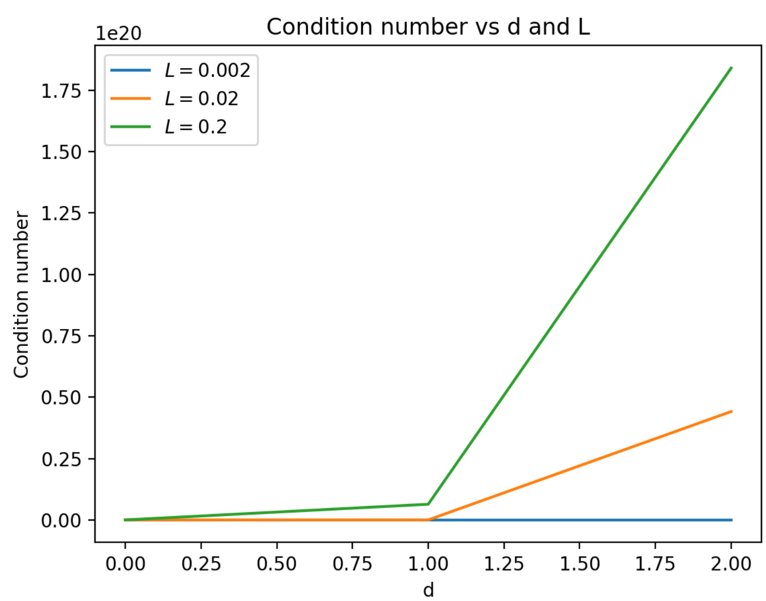

Figure 11: Condition number of the matrix M used in the linear equation vs d

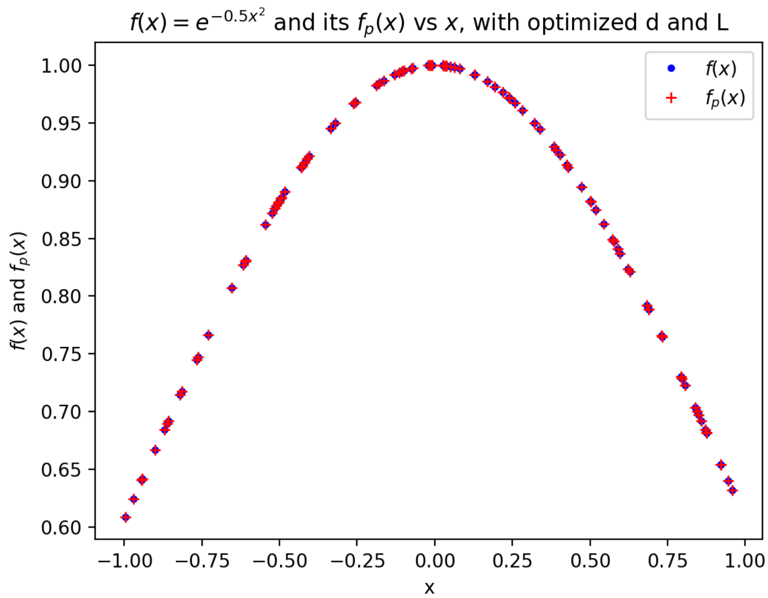

Figure 12: $f(x) = e^{-0.5x^2}$ and its $f_p(x)$ vs x