Analyse Numérique
Equations Non Linéaires

Simone Deparis

February 21, 2023

Contents

1 Equations non-linéaires

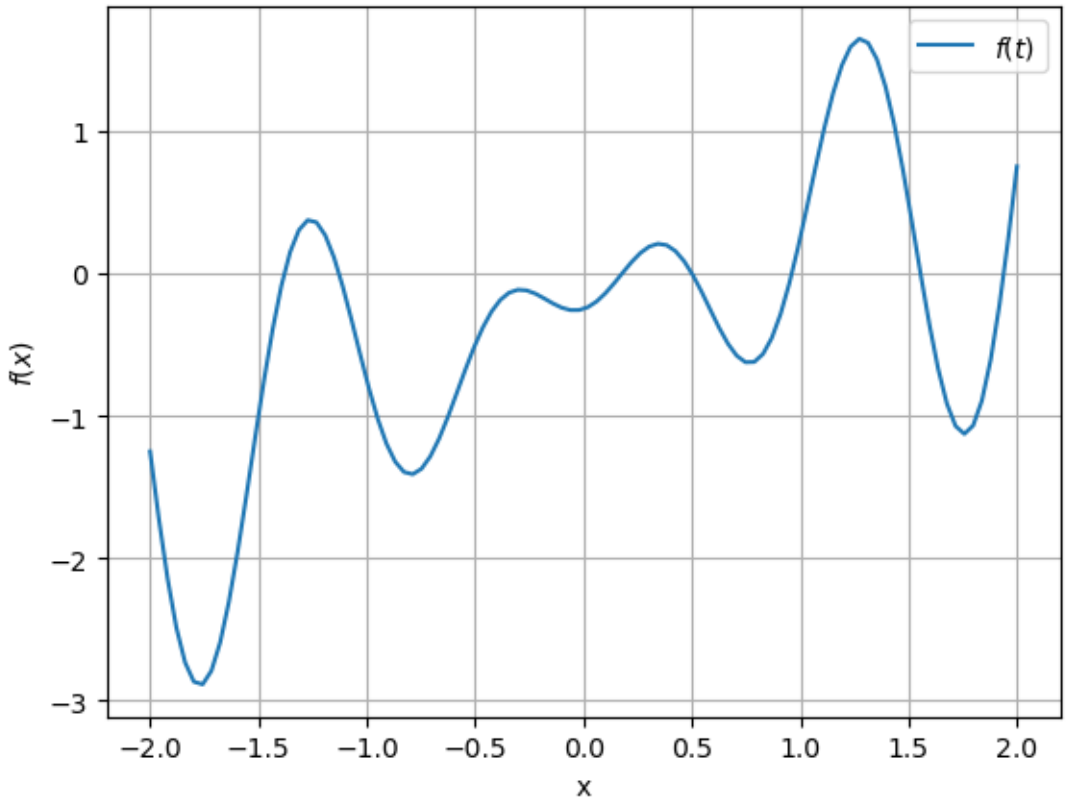
Objectif : trouver les zéros (ou racines) d'une fonction $f : [a, b] \to \mathbf{R}$:

$$\alpha \in [a, b] \ : \ f(\alpha) = 0$$

[1]:
```
# importing libraries used in this book
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
# defining the fonction that we want to interpolate
def f(x):
    return x*np.sin(x*2.*np.pi) + 0.5*x - 0.25

[a,b] = [-2,2]

# points used to plot the graph
z = np.linspace(a, b, 100)

plt.plot(z, f(z),'-')

# labels, title, legend
plt.xlabel('x'); plt.ylabel('$f(x)$'); #plt.title('data')
plt.legend(['$f(t)$'])
```

```
plt.grid(True)
plt.show()
```



1.1 Méthode de dichotomie ou bissection

Si f est continue et change de signe dans $[a, b]$, alors il existe au moins un α tel que $f(\alpha) = 0$.

On peut alors définir l'algorithme suivant :

$a^{(0)} = a$, $b^{(0)} = b$. Pour $k = 0, 1, ...$

1. $x^{(k)} = \frac{a^{(k)}+b^{(k)}}{2}$
2. si $f(x^{(k)}) = 0$, alors $x^{(k)}$ est le zéro cherché.

   Autrement:

   1. soit $f(x^{(k)})f(a^{(k)}) < k \Rightarrow$ et le zéro $\alpha \in [a^{(k)}, x^{(k)}]$.

   On pose $a^{(k+1)} = a^{(k)}$ et $b^{(k+1)} = x^{(k)}$

   2. soit $f(x^{(k)}f(b^{(k)}) < k \Rightarrow$ et le zéro $\alpha \in [x^{(k)}, b^{(k)}]$.

   On pose $a^{(k+1)} = x^{(k)}$ et $b^{(k+1)} = b^{(k)}$

1.1.1 Exercice

Ecrivez une fonction qui effectue l'algorithme de dichotomie ayant la structure suivante:

```
def bisection(a,b,f,tolerance,maxIterations) :
    # [a,b] interval of interest
    # f function
    # tolerance desired accuracy
    # maxIterations : maximum number of iteration
    # returns:
    # zero, residual, number of iterations
```

Ensuite testez-la pour trouver la racine de la fonction $f(x) = x\sin(2\pi x) + \frac{1}{2}x - \frac{1}{4}$ dans l'intervalle $[-1.5, 1]$.

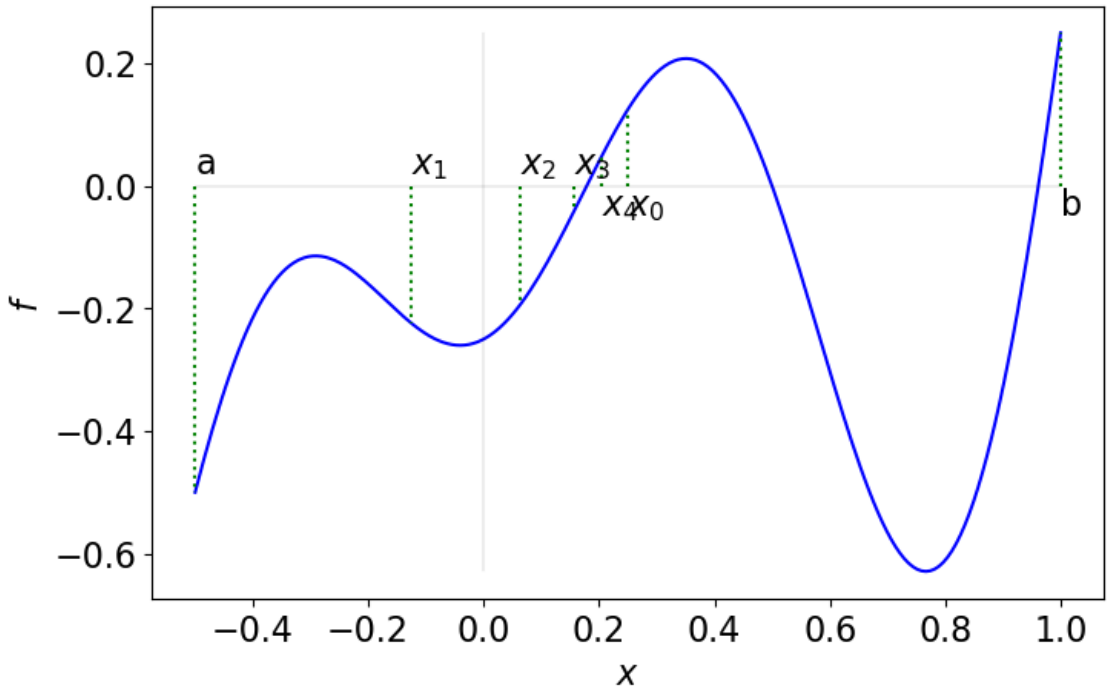
[3]:
```
def bisection(a,b,fun,tol,maxIterations) :
    # [a,b] interval of interest
    # fun function
    # tolerance desired accuracy
    # maxIterations : maximum number of iteration
    # returns:
    # zero, residual, number of iterations, x_sequence
    # x_sequence : the sequence computed by the fixed point iterations

    if (a >= b) :
        print(' b must be greater than a (b > a)')
        return 0,0,0,[0]

    # what we consider as "zero"
    eps = 1e-12

    # evaluate f at the endpoints
    fa = fun(a)
    fb = fun(b)

    if abs(fa) <  eps : # a is the solution
        zero = a
        esterr = fa
        k = 0
        return zero, esterr, k, [zero]

    if abs(fb) <  eps : # b is the solution
        zero = b
        esterr = fb
        k = 0
        return zero, esterr, k, [zero]

    if fa*fb > 0 :
```

```
        print(' The sign of FUN at the extrema of the interval must be␣
↪different')
        return 0,0,0,[0]



    # We want the final error to be smaller than tol,
    # i.e. k > log( (b-a)/tol ) / log(2) -1

    nmax = int(np.ceil(np.log( (b-a)/tol ) / np.log(2))) - 1


    # but nmax shall be smaller the the nmaximum iterations asked by the user
    if ( maxIterations < nmax ) :
        nmax = int(round(maxIterations))
        print('Warning: maxIterations is smaller than the minimum number of␣
↪iterations necessary to reach the tolerance wished');

    # vector of intermadiate approximations etc
    x = np.zeros(nmax+1)

    # initial error is the length of the interval.
    esterr  = (b - a)

    # do not need to store all the a^k and b^k, so I call them with a new␣
↪variable name:
    ak = a
    bk = b
    # the values of f at those points are fa and fk

    for k in range(nmax+1) :

        # approximate solution is midpoint of current interval
        x[k] =  (ak + bk) / 2
        fx = fun(x[k]);
        # error estimator is the half of the previous error
        esterr  = esterr / 2

        # if we found the solution, stop the algorithm
        if np.abs(fx) <  eps :
            # error is zero
            zero = x[k]
            esterr = 0;
            return zero, esterr, k, x

        if fx*fa < 0  :  # alpha is in (a,x)
            bk = x[k]
```

```
            fb = fx
        elif fx*fb < 0 : # alpha is in (x,b)
            ak = x[k]
            fa = fx
        else  :
            error('Algorithm not operating correctly')


    zero = x[k];

    if esterr > tol :
        print('Warning: bisection stopped without converging to the desired␣
 ↪tolerance because the maximum number of iterations was reached');

    return zero, esterr, k, x
```

[4]:
```
from NonLinearEquationsLib import plotBisectionIterations

[a,b] = [-.5,1]
tol = 1e-10
maxIter = 4
zero, esterr, k, x = bisection(a,b,f,tol,maxIter)

plt.rcParams.update({'font.size': 16})
plt.figure(figsize=(8, 5))

plotBisectionIterations(a,b,f,x)

# plt.savefig('Bisection-iterations.png', dpi=600)

print(f'The estimated root is {zero:2.12f}, the estimated error {esterr:1.3e}␣
 ↪and the residual is {f(zero):1.3e}, after {k} iterations')
```

```
Warning: maxIterations is smaller than the minimum number of iterations
necessary to reach the tolerance wished
Warning: bisection stopped without converging to the desired tolerance because
the maximum number of iterations was reached
The estimated root is 0.203125000000, the estimated error 4.688e-02 and the
residual is 4.594e-02, after 4 iterations
```



[ ]:

2 Equations non-linéaires

Objectif : trouver les zéros (ou racines) d'une fonction $f : [a, b] \to \mathbf{R}$:

$$\alpha \in [a, b] \ : \ f(\alpha) = 0$$

[5]:
```
# importing libraries used in this book
import numpy as np
import matplotlib.pyplot as plt
```

2.1 Méthode de Newton

Soit $f : \mathbb{R} \to \mathbb{R}$ une fonction différentiable.

Soit $x^{(0)}$ un point donné. On considère l'équation de la droite $y(x)$ qui passe par le point $(x^{(k)}, f(x^{(k)}))$ et qui a comme pente $f'(x^{(k)})$,

$$y(x) = f'(x^{(k)})(x - x^{(k)}) + f(x^{(k)}).$$

On définit $x^{(k+1)}$ comme étant le point où cette droite intersecte l'axe x, c'est-à-dire $y(x^{(k+1)}) = 0$. On en déduit que :

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})}, \quad k = 0, 1, 2 \ldots .$$

[6]:
```
# importing libraries used in this book
import numpy as np
import matplotlib.pyplot as plt
```

Exercice (5, Série 1) On cherche les zéros de la fonction

$$f(x) = \frac{1}{2}\sin\left(\frac{\pi x}{2}\right) + 1 - x\ .$$

1. Vérifiez qu'il y a au moins un zéro α dans l'intervalle $[0, 2]$.
2. Ecrivez la méthode de Newton pour trouver le zéro α de la fonction $f(x)$ et calculez la première itération à partir de la valeur initiale $x^{(0)} = 1$.
3. Calculez les zéros α de la fonction f avec la méthode de Newton (fonction `newton` que vous devrez écrire)

```
def  Newton( F, dF, x0, tol, nmax ) :
    # NEWTON Find the zeros of a nonlinear equations.
    # NEWTON(F,DF,X0,TOL,NMAX) tries to find the zero X of the
    # continuous and differentiable function F nearest to X0 using
    # the Newton method. DF is a function which take X and return the derivative of F.
    # If the search fails an error message is displayed.
    #
    # returns the value of the
    # residual R in X,the number of iterations N required for computing X and
    # INC the increments computed by Newton.

    return x, r, n, inc
```

Choisissez $x^{(0)} = 1$ comme point de départ pour la méthode et utilisez une tolérance $tol = 10^{-4}$ sur la valeur absolue de l'incrément entre deux itérations successives $|x^{(k+1)} - x^{(k)}|$.

Dans le cas de la méthode de Newton, l'incrément est une bonne approximation de l'erreur.

[7]:
```
def  Newton( F, dF, x0, tol, nmax ) :
    '''
    NEWTON Find the zeros of a nonlinear equations.
    NEWTON(F,DF,X0,TOL,NMAX) tries to find the zero X of the
    continuous and differentiable function F nearest to X0 using
    the Newton method. DF is a function which take X and return the derivative␣
 ↪of F.
    If the search fails an error message is displayed.

    Outputs : [x, r, n, inc, x_sequence]
    x : the approximated root of the function
    r : the absolute value of the residualin X
    n : the number of iterations N required for computing X and
    inc : the increments computed by Newton.
    x_sequence : the sequence computed by Newton
```

```
    '''

    # Initial values
    n = 0
    xk = x0

    # initialisation of loop components
    # increments (in abs value) at each iteration
    inc = []
    # in case we wish to plot the sequence
    x = [x0]

    # diff : last increment,
    diff = tol + 1 # initially set larger then tolerance

    # Loop until tolerance is reached
    while ( diff >= tol and n <= nmax ) :
        # Newton iteration
        deltax  = F(xk) / dF(xk)
        xk1 = xk - deltax

        # increments
        diff = np.abs(deltax)
        inc.append (diff)

        # prepare the next loop
        n = n + 1
        xk = xk1
        x.append(xk)

    # Final residual
    rk1 = np.abs(F(xk1))

    # Warning if not converged
    if n > nmax :
        print('Newton stopped without converging to the desired tolerance ')
        print('because the maximum number of iterations was reached')

    return xk1, rk1, n, inc, np.array(x)
```

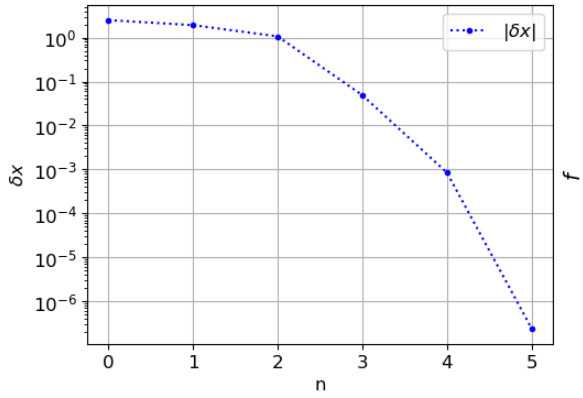
[8]:
```
f  = lambda x : 0.5*np.sin(np.pi*x/2)+1-x
df = lambda x : 0.25*np.pi*np.cos(np.pi*x/2)-1

x0   = 1
tol  = 1e-4
nmax = 10
```

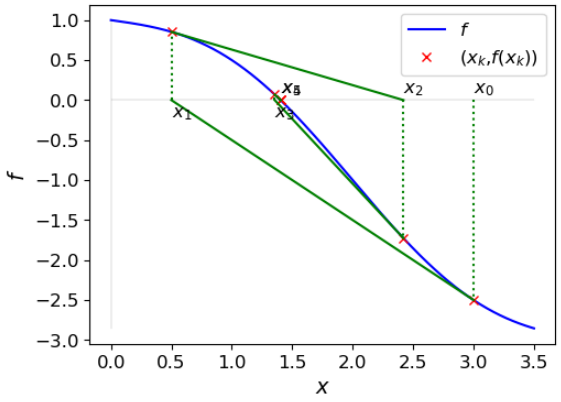
```
zero, residual, niter, inc, x = Newton(f, df, x0, tol, nmax)

print(f'The zero computed is {zero:1.4f}')
print(f'Newton stoppedconverged in {niter} iterations');
print(f'with a residual of {residual:1.4e}.\n');
```

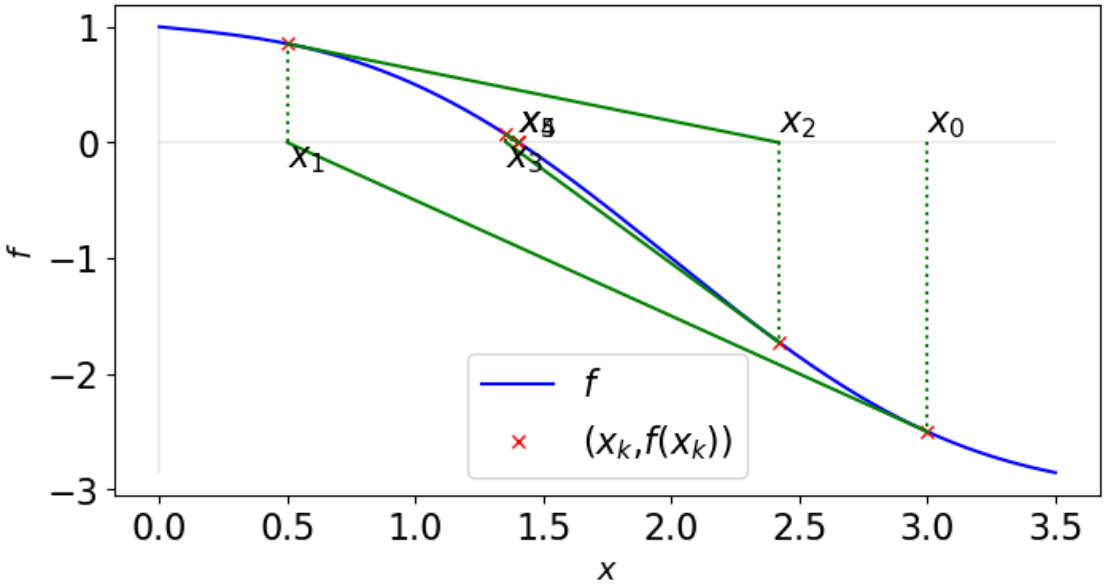
```
The zero computed is 1.4031
Newton stoppedconverged in 4 iterations
with a residual of 3.3120e-12.
```

[9]:
```
from NonLinearEquationsLib import plotNewtonIterations
```

[10]:
```
[a,b] = [0,3.5]
x0   = 3
zero, residual, niter, inc, x = Newton(f, df, x0, tol, nmax)

# Rezise plots, which are usually too small
plt.figure(figsize=(12, 4))
plt.rcParams.update({'font.size': 12})

# Subplot 1 over 2, 1st one
plt.subplot(121)

#plt.plot(range(MaxIterations), RelativeError, 'b:.')
plt.plot(range(niter), inc, 'b:.')

plt.xlabel('n'); plt.ylabel('$\\delta x$');
plt.grid(True)
#plt.xscale('log')
plt.yscale('log')
plt.legend(['$|\\delta x|$'])


# Subplot 1 over 2, 2nd one
plt.subplot(122)

plotNewtonIterations (a,b,f,x,200)

plt.show()

# Rezise plots, which are usually too small
plt.figure(figsize=(8, 4))
plt.rcParams.update({'font.size': 16})

plotNewtonIterations (a,b,f,x,200)
# plt.savefig('Newton-iterations.png', dpi=600)
plt.show()
```

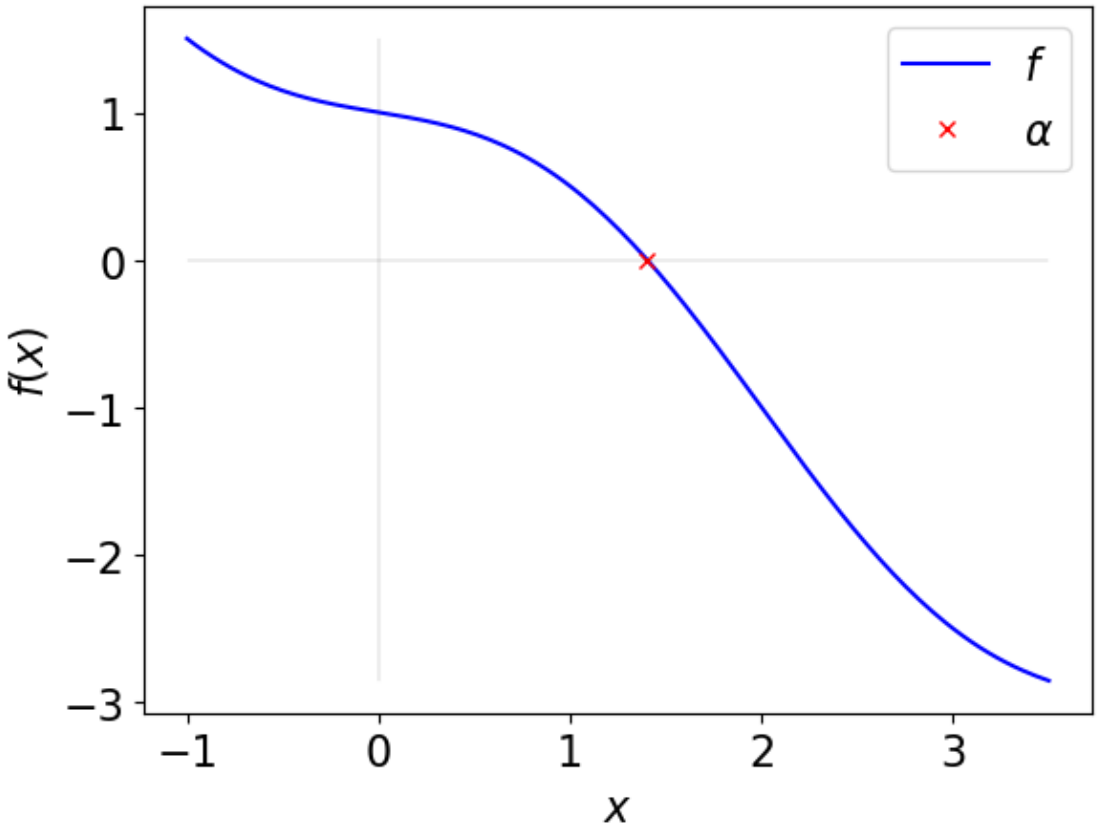






[11]:
```
[a,b] = [-1,3.5]
z = np.linspace(a,b,200)
plt.plot(z,f(z), 'b-', x[6],f(x[6]), 'rx')
plt.ylabel('$f(x)$'); plt.xlabel('$x$');

# Plot the x,y-axis
plt.plot([a,b], [0,0], 'k-',linewidth=0.1)
plt.plot([0,0], [np.min(f(z)),np.max(f(z))], 'k-',linewidth=0.1)

plt.legend(['$f$','$\\alpha$'])
# plt.savefig('Newton-fx-alpha.png', dpi=600)
```

```
plt.show()
```



2.2 Critères d'arrêt

On a l'estimation

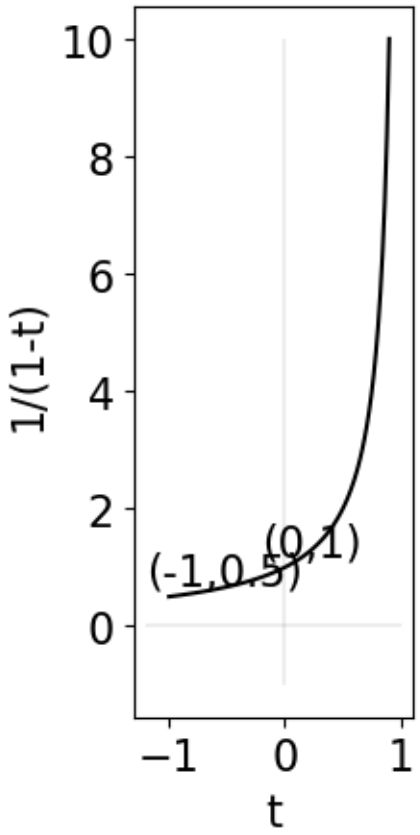
$$e^{(k)} = \frac{1}{\left(1-\phi'(\xi^{(k)})\right)}(x^{(k+1)} - x^{(k)}). = \gamma(\phi'(\xi^{(k)}))(x^{(k+1)} - x^{(k)})$$

On cherche à obtenir $|e^{(k)}| \approx \epsilon$ (une tolérance choisie).

On trace un graphe de la fonction $\gamma(t) = \frac{1}{1-t}$

[12]:
```
# Graph of the fonction $f(t)=1/(1-t)$
N = 100
a,b = [-1,0.9]
z = np.linspace(a,b,N)
f = lambda x : 1/(1-x)

plt.subplot(1,3,1)
plt.plot(z,f(z),'k-')
```

```
plt.annotate("(-1,0.5)", (-1.2, 0.7))

plt.annotate("(0,1)", (-0.2, 1.2))


plt.plot([-1.2,1], [0,0], 'k-',linewidth=0.1)
plt.plot([0,0], [-1,10], 'k-',linewidth=0.1)


plt.xlabel('t'); plt.ylabel('1/(1-t)');
# Plot the x,y-axis

plt.show()
```



$$\left.\begin{array}{l}\text{Si } t<0,\ \gamma(t)\in[\frac{1}{2},1]\\ \text{Si } t\approx 0,\ \gamma(t)\approx 1\\ \text{Si } t\to 1,\ \gamma(t)\to\infty\end{array}\right\}\quad \begin{array}{lll}|e^{(k)}|\lesssim|x^{(k+1)}-x^{(k)}| & \Rightarrow & |x^{(k+1)}-x^{(k)}|<\epsilon\\ \phi'(\xi^{(k)})\approx\phi'(x^{(k)}) & \Rightarrow & |x^{(k+1)}-x^{(k)}|<\epsilon(1-\phi'(x^{(k)}))\end{array}$$

[ ]:

3 Methode de point fixe

3.0.1 Exercice (1, Série 3)

On considère les méthodes de point fixe $x^{(n+1)} = g_i(x^{(n)}) \quad (i = 1, 2, 3)$ avec:

$$g_1(x^{(n)}) = \frac{1}{2}e^{x^{(n)}/2}, \qquad g_2(x^{(n)}) = -\frac{1}{2}e^{x^{(n)}/2}, \qquad g_3(x^{(n)}) = 2\ln(2x^{(n)}),$$

dont les fonctions d'itération $g_i(x)$ sont visualisées sur la figure plus bas

1. Pour chaque point fixe $\bar{x}$ de la fonction d'itération g_i $(i = 1, 2, 3)$, on suppose d'avoir choisi une valeur initiale $x^{(0)}$ proche de $\bar{x}$. Etudiez si la méthode converge vers $\bar{x}$.
2. Pour chaque fonction d'itération g_i, déterminez graphiquement pour quelles valeurs initiales $x^{(0)}$ la méthode de point fixe correspondante converge et vers quel point fixe.
3. Montrez que si $\bar{x}$ est un point fixe de la fonction g_i $(i = 1, 2, 3)$, alors il est aussi un zéro de la fonction $f(x) = e^x - 4x^2$ (dont le comportement est tracé sur la dernière figure).
4. Comment peut-on calculer les zéros de f?

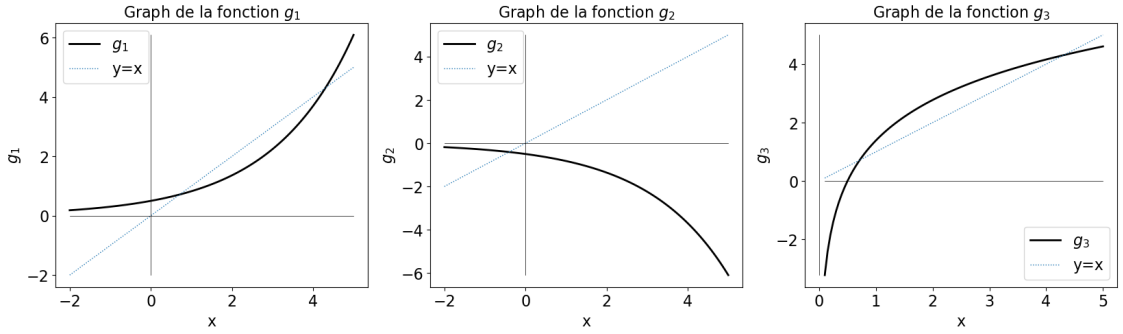
L'exercice 2 est à faire sur papier, ici une indication par ordinateur

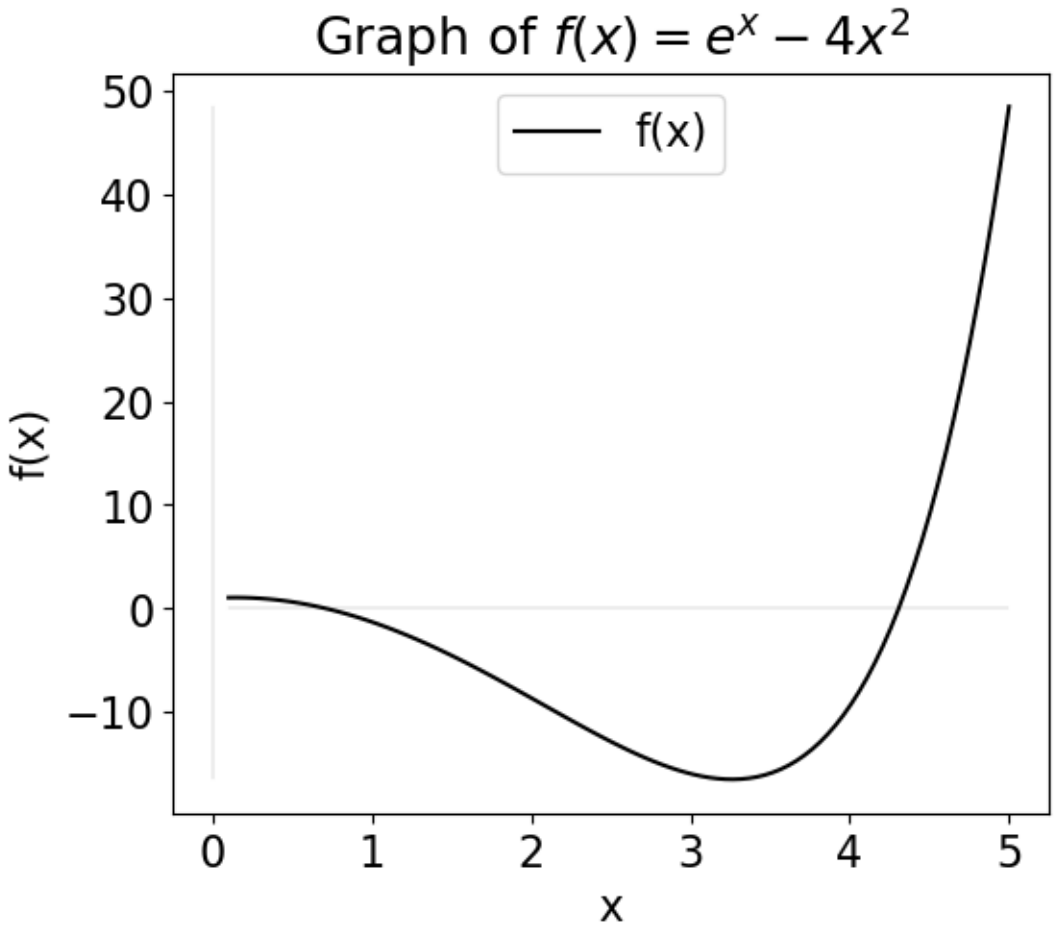
[13]:
```
# importing libraries used in this book
import numpy as np
import matplotlib.pyplot as plt
```

[14]:
```
from NonLinearEquationsLib import plotPhi, FixedPoint, plotPhiIterations
```

[15]:
```
plt.rcParams['figure.figsize'] = [20, 5]

plt.subplot(1,3,1)
[a,b] = [-2,5]
phi1 = lambda x : np.exp(x/2)/2
plotPhi(a,b,phi1,'$g_1$')

plt.subplot(1,3,2)
[a,b] = [-2,5]
phi2 = lambda x : - np.exp(x/2)/2
plotPhi(a,b,phi2,'$g_2$')

plt.subplot(1,3,3)
[a,b] = [1e-1,5]
phi3 = lambda x : 2*np.log(2*x)
plotPhi(a,b,phi3,'$g_3$')

plt.show()

# Graph of the fonction $f(x)=e^x-4x^2$
N = 100
z = np.linspace(a,b,N)
```

```
f = lambda x : np.exp(x) - 4*x*x

plt.subplot(1,3,1)
plt.plot(z,f(z),'k-')

plt.xlabel('x'); plt.ylabel('f(x)');
# Plot the x,y-axis
plt.plot([a,b], [0,0], 'k-',linewidth=0.1)
plt.plot([0,0], [np.min(f(z)),np.max(f(z))], 'k-',linewidth=0.1)
plt.legend(['f(x)'])
plt.title('Graph of $f(x)=e^x-4x^2$')

plt.show()
```





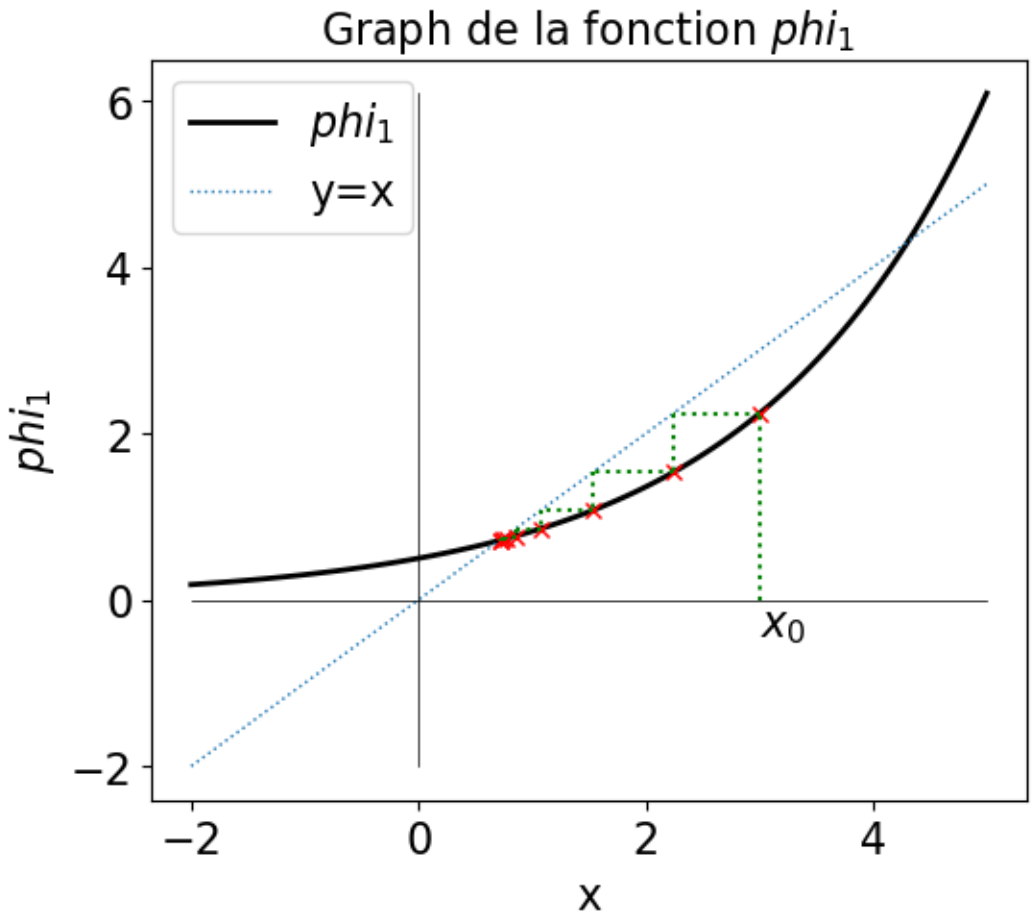
Partie 1 *Pour chaque point fixe $\bar{x}$ de la fonction d'itération g_i $(i = 1, 2, 3)$, on suppose choisir une valeur initiale $x^{(0)}$ proche de $\bar{x}$. Etudiez si la méthode converge vers $\bar{x}$.*

On va utiliser la fonction `FixedPoint` qui se trouve dans `NonLinearEquationsLib.py`

```
def  FixedPoint( phi, x0, a, b tol, nmax ) :
    '''
    FixedPoint Find the fixed point of a function by iterative iterations
    FixedPoint( PHI,X0,a,b, TOL,NMAX) tries to find the fixedpoint X of the a
    continuous function PHI nearest to X0 using
    the fixed point iterations method.
    [a,b] : if the iterations exit the interval, the method stops
    If the search fails an error message is displayed.

    Outputs : [x, r, n, inc, x_sequence]
    x : the approximated fixed point of the function
    r : the absolute value of the residual in X : |phi(x) - x|
    n : the number of iterations N required for computing X and
    x_sequence : the sequence computed by Newton
```

```
    ...
    return xk1, rk1, n, np.array(x)
    '''
```

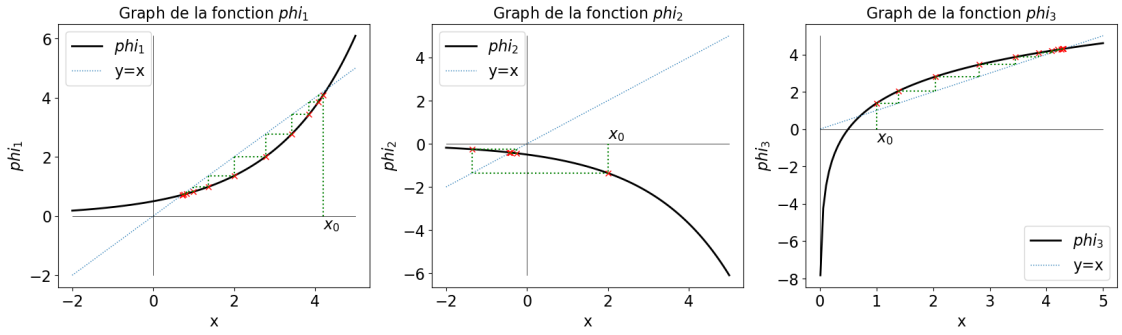
[16]:
```
tol  = 1e-2
nmax = 10

# Choose fonction phi
[a,b] = [-2,5]
phi = phi1
label = '$phi_1$'

# Initial Point
x0   = 3
zero, residual, niter, x = FixedPoint(phi, x0, a,b, tol, nmax)

plt.subplot(131)
plotPhi (a,b,phi,label)
plt.plot(x,phi(x), 'rx')

# plot the graphical interpretation of the Fixed Point method
plotPhiIterations(x)

plt.show()
```



[17]:
```
tol  = 1e-2
nmax = 10

# Choose fonction phi
intervals = [ [-2,5] , [-2,5] , [1e-2,5] ]
phiFunctions = [phi1, phi2, phi3]
labels = ['$phi_1$', '$phi_2$', '$phi_3$']
# Initial Points
initialPoints   = [4.2,2,1]

alpha = np.empty(3)

for k in range(3) :
    phi = phiFunctions[k]; x0 = initialPoints[k]
    a = intervals[k][0]; b = intervals[k][1]
    label = labels[k]
```

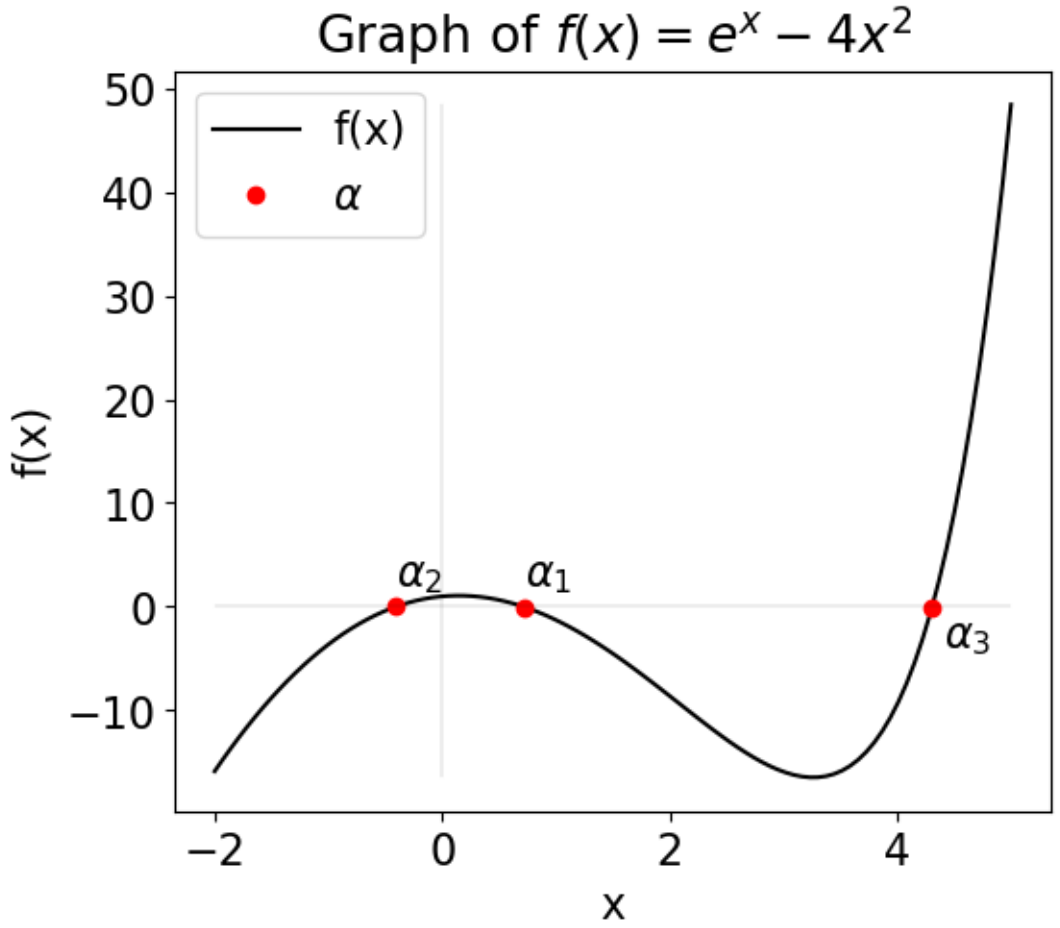
```
    alpha[k], residual, niter, x = FixedPoint(phi, x0, a,b, tol, nmax)

    plt.subplot(1,3,k+1)

    plotPhi (a,b,phi,label)
    plt.plot(x,phi(x), 'rx')

    # plot the graphical interpretation of the Fixed Point method
    plotPhiIterations(x)



plt.show()
```

```
FixexPoint stopped without converging to the desired tolerance
because the maximum number of iterations was reached
FixexPoint stopped without converging to the desired tolerance
because the maximum number of iterations was reached
```



[18]:
```
# Graph of the fonction $f(x)=e^x-4x^2$
N = 100
a,b = [-2,5]
z = np.linspace(a,b,N)
f = lambda x : np.exp(x) - 4*x*x

plt.subplot(1,3,1)
plt.plot(z,f(z),'k-')

# Solutions found:
plt.plot(alpha,f(alpha),'ro')
plt.annotate("$\\alpha_1$", (alpha[0], 2))
plt.annotate("$\\alpha_2$", (alpha[1], 2))
plt.annotate("$\\alpha_3$", (alpha[2]+0.1, -4))

plt.xlabel('x'); plt.ylabel('f(x)');
```

```
# Plot the x,y-axis
plt.plot([a,b], [0,0], 'k-',linewidth=0.1)
plt.plot([0,0], [np.min(f(z)),np.max(f(z))], 'k-',linewidth=0.1)
plt.legend(['f(x)','$\\alpha$'])
plt.title('Graph of $f(x)=e^x-4x^2$')

plt.show()
```



[ ]: