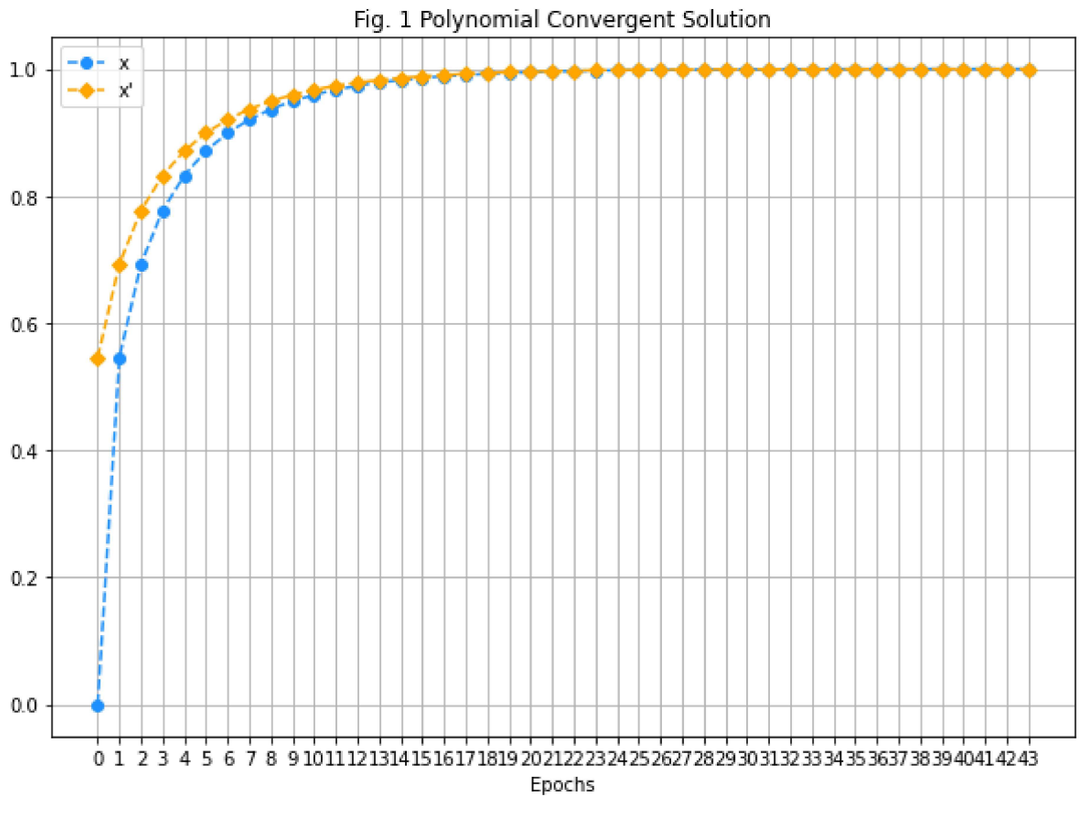
```
import numpy as np
import matplotlib.pyplot as plt
```

Brute force (In terms of X)

```
func = lambda x: x**3-6*x**2+11*x-6
f1 = lambda x: (-x**3+6*x**2+6)/11
f2 = lambda x: ((x**3+11*x-6)/6)**(1/2)
f3 = lambda x: (6*x**2-11*x+6)**(1/3)
funcs5 = [f1,f2,f3]
epochs0 = 1000000
```

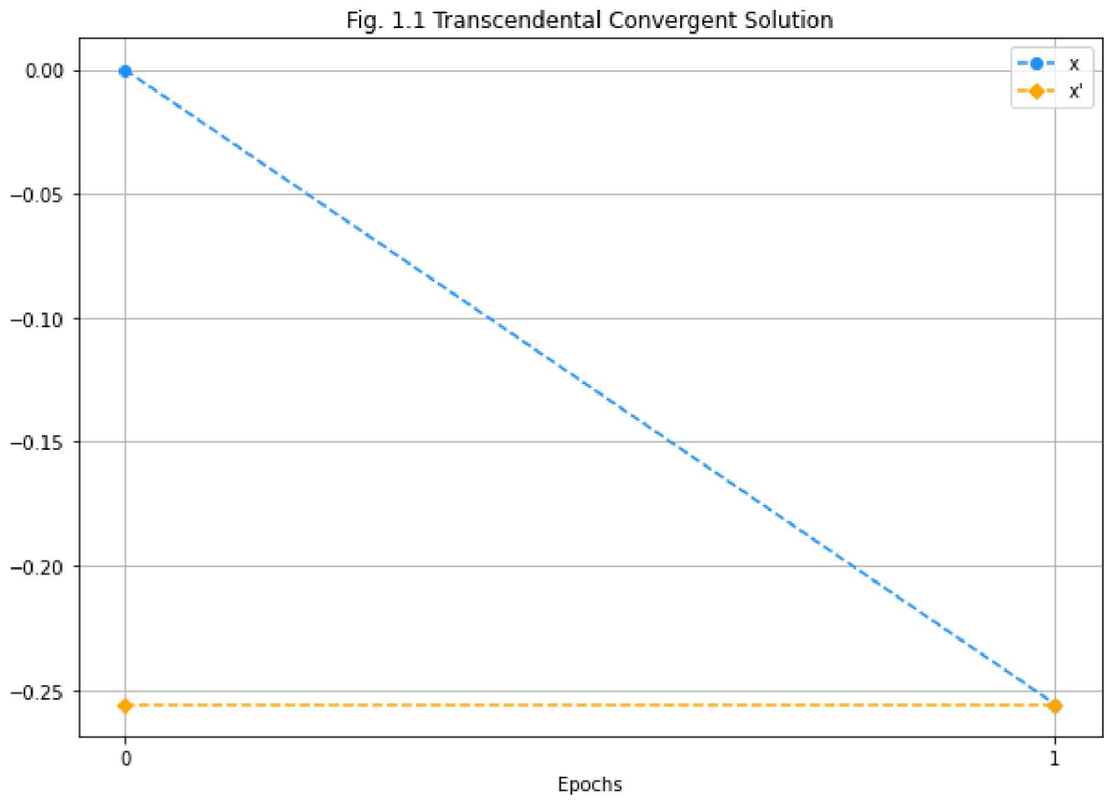
```
def b_forcex(funcs,epochs,x0 = 0):
  history_x = []
  history_xp = []
  for func in funcs:
    x0=0
    for epoch in range(epochs):
      x_prime = func(x0)
      history_x.append(x0)
      history_xp.append(x_prime)
      if np.allclose(x0, x_prime):
        break
      x0 = x_prime
    return history_x, history_xp
history_x,history_xp =b_forcex(funcs5,epochs0)
x_range = np.arange(len(history_x),dtype=int)
plt.figure(figsize=(10,7))
plt.plot(x_range, history_x, 'o--',color='dodgerblue', label='x')
plt.plot(x_range, history_xp,'D--', color='orange', label='x\'')
plt.title("Fig. 1 Polynomial Convergent Solution")
plt.xlabel("Epochs")
plt.xticks(np.arange(min(x_range), max(x_range)+1, 1.0))
plt.legend()
plt.grid()
plt.show()
```



It can be seen in the graph that in about 13 epochs, the x value (dodger blue) and the computed value (orange) meets. This means that the polynomial convergent solution using the developed brute force (in terms of X) algorithm was excellent since the values met at some point.

```
sample2 = lambda x: np.sin(2*x)-np.cos(2*x)
f1 = lambda x: np.arcsin(np.cos(16))/5
# f1 = lambda x: np.arcsin(np.cos(2*x))/2
# f2 = lambda x: np.arccos(np.sin(2*x))/2
funcs4 = [f1]
no_roots = len(funcs4)
```

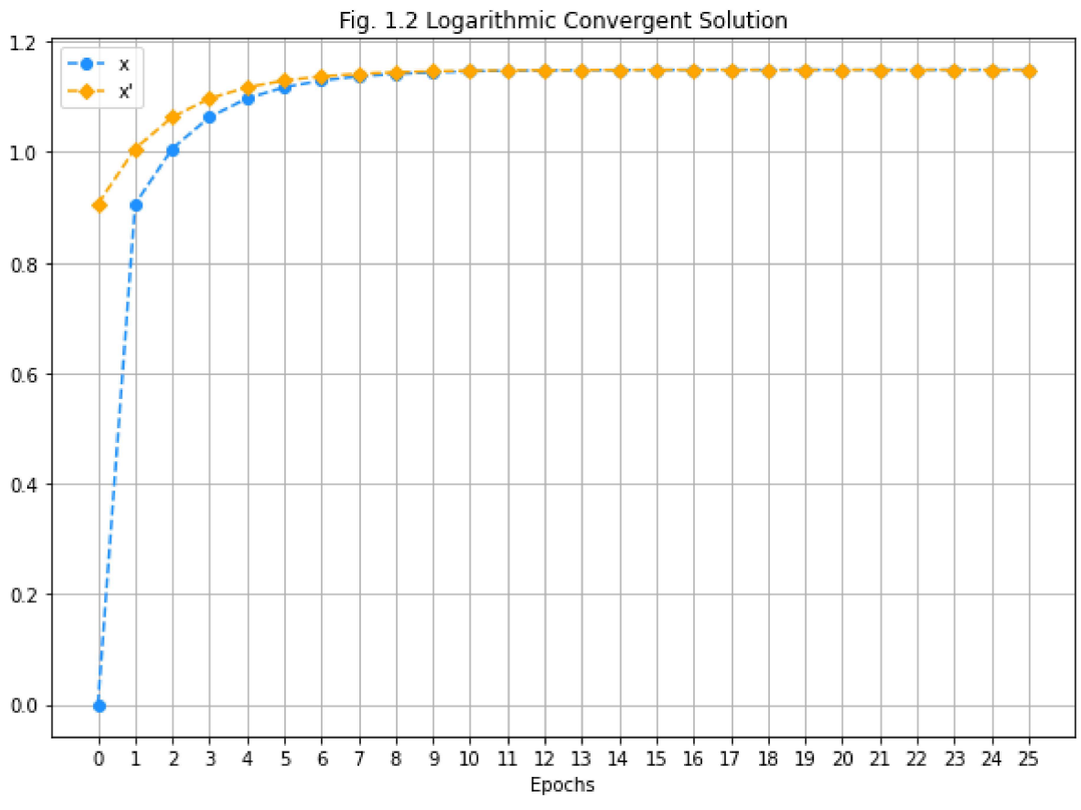
```
def b_forcex(funcs,no_roots,epochs,x0 = 0):
  roots = []
  history_x = []
  history_xp = []
  for func in funcs:
    x0=0
    for epoch in range(epochs):
      x_prime = func(x0)
      history_x.append(x0)
      history_xp.append(x_prime)
      if np.allclose(x0, x_prime,1e-06):
        break
      x0 = x_prime
    return history_x, history_xp
history_x,history_xp = b_forcex(funcs4,no_roots,epochs0)
x_range = np.arange(len(history_x),dtype=int)
plt.figure(figsize=(10,7))
plt.plot(x_range, history_x, 'o--',color='dodgerblue', label='x')
plt.plot(x_range, history_xp,'D--', color='orange', label='x\'')
plt.title("Fig. 1.1 Transcendental Convergent Solution")
plt.xlabel("Epochs")
plt.xticks(np.arange(min(x_range), max(x_range)+1, 1.0))
plt.legend()
plt.grid()
plt.show()
```



It can be seen in the graph that in just 1 epoch, the x value (dodger blue) and the computed value (orange) meets. This means that the transcendental convergent solution using the developed brute force (in terms of X) algorithm was excellent since the values met at some point

```
sample6 = lambda x: np.log(4*x**2-3*x+1)*(x**5-2)
f1 = lambda x: ((3*x**6-x**5+8*x**2-6*x+2)/4)**(1/7)
f2 = lambda x: ((4*x**7+x**5-8*x**2-6*x-2)/3)**(1/6)
f3 = lambda x: (-4*x**7+3*x**6+8*x**2-6*x+2)**(1/5)
f4 = lambda x: ((4*x**7-3*x**6+x**5+6*x-2)/8)**(1/2)
f5 = lambda x: (-4*x**7+3*x**6-x**5+8*x**2+2)/6
funcs6 = [f1,f3,f4,f5]
no_roots = len(funcs6)
```

```
def b_forcex(funcs,no_roots,epochs,x0 = 0):
  roots = []
  history_x = []
  history_xp = []
  for func in funcs:
    x0=0
    for epoch in range(epochs):
      x_prime = func(x0)
      history_x.append(x0)
      history_xp.append(x_prime)
      if np.allclose(x0, x_prime,1e-06):
        break
      x0 = x_prime
    return history_x, history_xp
history_x,history_xp = b_forcex(funcs6,no_roots,epochs0)
x_range = np.arange(len(history_x),dtype=int)
plt.figure(figsize=(10,7))
plt.plot(x_range, history_x, 'o--',color='dodgerblue', label='x')
plt.plot(x_range, history_xp,'D--', color='orange', label='x\'')
plt.title("Fig. 1.2 Logarithmic Convergent Solution")
plt.xlabel("Epochs")
plt.xticks(np.arange(min(x_range), max(x_range)+1, 1.0))
plt.legend()
plt.grid()
```



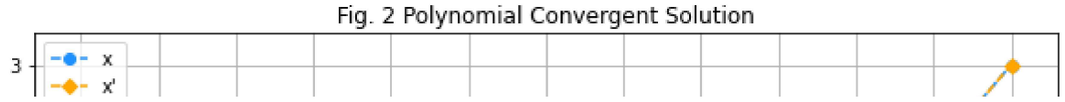
It can be seen in the graph that in 7-8 epochs, the x value (dodger blue) and the computed value (orange) meets. This means that the logarithmic convergent solution using the developed brute

force (in terms of X) algorithm was excellent since the values met at some point.

Newton-Raphson Method

```
def derivative(f,x,dx = 1e-6):
  diff = f(x+dx)-f(x-dx)
  return diff/(2*dx)

def newton(func,n_roots,epochs, tol = 1.0e-05,inits = np.arange(-5,5)):
  history_x =[]
  history_x_prime = []
  for init in inits:
    x=init
    for epoch in range(epochs):
      f_prime = derivative(func,x)
      x_new = x - (func(x)/f_prime)
      history_x.append(x)
      history_x_prime.append(x_new)
      if np.allclose(x, x_new, tol):
        break
      x = x_new
  return history_x,history_x_prime,epoch

func = lambda x: x**3-6*x**2+11*x-6 #np.sin(2*x)-np.cos(2*x)
history_x,history_x_prime,epoch = newton(func,n_roots = 5,epochs = 2, tol = 1.0e-05,inits = r


history_x = history_x[0:]
history_x_prime = history_x_prime[0:]
x_range = np.arange(len(history_x),dtype=int)

plt.figure(figsize=(10,7))
plt.plot(x_range, history_x, 'o--',color='dodgerblue', label='x')
plt.plot(x_range, history_x_prime,'D--', color='orange', label='x\'')

plt.title("Fig. 2 Polynomial Convergent Solution")
plt.xlabel("Epochs")
plt.xticks(np.arange(min(x_range), max(x_range)+1, 1.0))
plt.legend()
plt.grid()
plt.show()
```

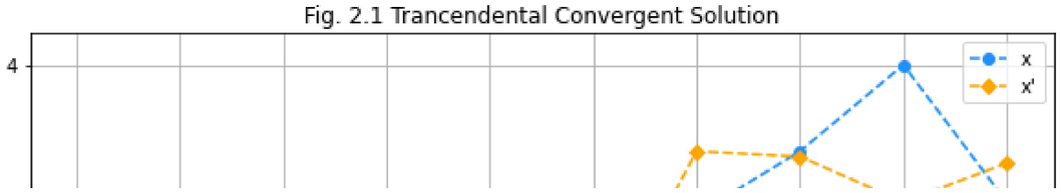
Fig. 2 Polynomial Convergent Solution



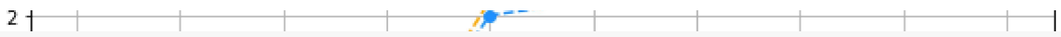
It can be seen in the graph that in 10 epoch, the x value (dodger blue) and the computed value (orange) meets. This means that the polynomial convergent solution using the developed newton-raphson method was excellent since the values met at some point.

```
def derivative(f,x,dx = 1e-6):
  diff = f(x+dx)-f(x-dx)
  return diff/(2*dx)

def newton(func,n_roots,epochs, tol = 1.0e-05,inits = np.arange(-5,5)):
  history_x =[]
  history_x_prime = []
  for init in inits:
    x=init
    for epoch in range(epochs):
      f_prime = derivative(func,x)
      x_new = x - (func(x)/f_prime)
      history_x.append(x)
      history_x_prime.append(x_new)
      if np.allclose(x, x_new, tol):
        break
      x = x_new
  return history_x,history_x_prime,epoch

func = lambda x: np.sin(5*x)-np.cos(2**4)
history_x,history_x_prime,epoch = newton(func,n_roots = 5,epochs = 2, tol = 1.0e-05,inits = r

history_x = history_x[0:]
history_x_prime = history_x_prime[0:]
x_range = np.arange(len(history_x),dtype=int)

plt.figure(figsize=(10,7))
plt.plot(x_range, history_x, 'o--',color='dodgerblue', label='x')
plt.plot(x_range, history_x_prime,'D--', color='orange', label='x\'')

plt.title("Fig. 2.1 Trancendental Convergent Solution")
plt.xlabel("Epochs")
plt.xticks(np.arange(min(x_range), max(x_range)+1, 1.0))
plt.legend()
plt.grid()
plt.show()
```

Fig. 2.1 Trancendental Convergent Solution

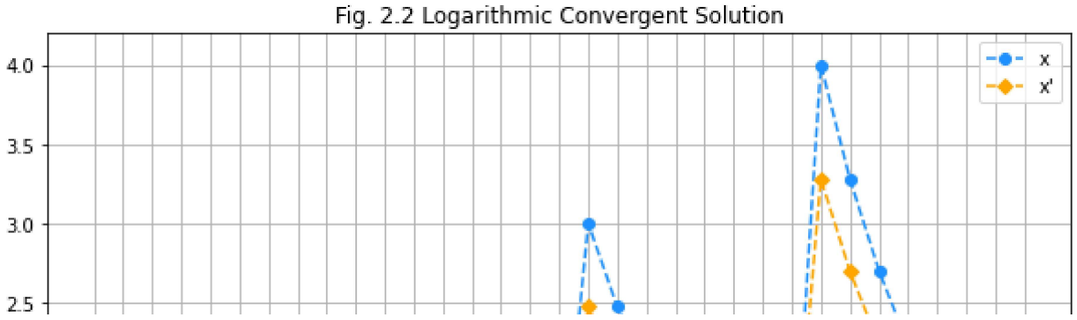


It can be seen in the graph that in 2 epochs, the x value (dodger blue) and the computed value (orange) meets. This means that the transcendental convergent solution using the developed newton-raphson method was excellent since the values met at some point.




```
def derivative(f,x,dx = 1e-6):
  diff = f(x+dx)-f(x-dx)
  return diff/(2*dx)

def newton(func,n_roots,epochs, tol = 1.0e-05,inits = np.arange(-5,5)):
  history_x =[]
  history_x_prime = []
  for init in inits:
    x=init
    for epoch in range(epochs):
      f_prime = derivative(func,x)
      x_new = x - (func(x)/f_prime)
      history_x.append(x)
      history_x_prime.append(x_new)
      if np.allclose(x, x_new, tol):
        break
      x = x_new
  return history_x,history_x_prime,epoch

func = lambda x: np.log(4*x**2-3*x+1)*(x**5-2)
history_x,history_x_prime,epoch = newton(func,n_roots = 5,epochs = 8, tol = 1.0e-05,inits = r


history_x = history_x[0:]
history_x_prime = history_x_prime[0:]
x_range = np.arange(len(history_x),dtype=int)

plt.figure(figsize=(10,7))
plt.plot(x_range, history_x, 'o--',color='dodgerblue', label='x')
plt.plot(x_range, history_x_prime,'D--', color='orange', label='x\'')

plt.title("Fig. 2.2 Logarithmic Convergent Solution")
plt.xlabel("Epochs")
plt.xticks(np.arange(min(x_range), max(x_range)+1, 1.0))
plt.legend()
plt.grid()
plt.show()
```

Fig. 2.2 Logarithmic Convergent Solution

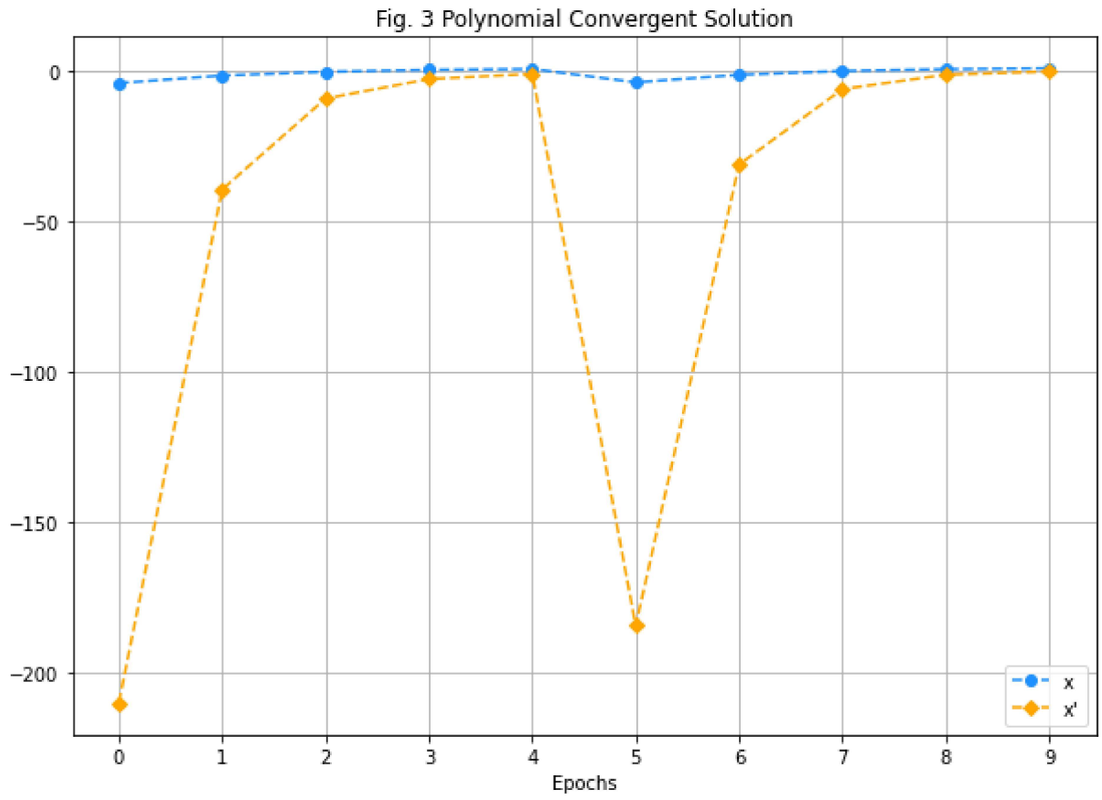


It can be seen in the graph that in 8 epochs, the x value (dodger blue) and the computed value (orange) meets. This means that the logarithmic convergent solution using the developed newton-raphson method was excellent since the values met at some point.

Bisection Method


```
def bisection(func,i1,i2,n_roots,epochs,n_move,tol = 1.0e-06):
  fpoint = []
  spoint = []
  history_x =[]
  history_x_prime = []

  for i in range(n_move): #Moving intervals
    i1+=0.25
    i2+=0.25
    fpoint.append(i1)
    spoint.append(i2)

  for (i1,i2) in zip (fpoint,spoint):
    y1, y2 = func(i1), func(i2)
    root = None
    end_bisect = 0
    if np.sign(y1) == np.sign(y2):
      pass #Root are not in this interval
    else:
      for bisect in range(epochs):
        midp = np.mean([i1,i2])
        y_mid = func(midp)
        y1 = func(i1)
        history_x.append(i1)
        history_x_prime.append(y1)
        if np.allclose(0,y1, tol):
          root = i1
          break
        if np.sign(y1) != np.sign(y_mid): #root is in first-half interval
          i2 = midp
        else: #root is in second-half interval
          i1 = midp

  return history_x,history_x_prime

func = lambda x: x**3-6*x**2+11*x-6

history_x,history_x_prime = bisection(func,i1 = -5,i2 = 0,n_roots = 5,epochs = 5,n_move = 5,t

history_x = history_x[0:]
history_x_prime = history_x_prime[0:]
x_range = np.arange(len(history_x),dtype=int)

plt.figure(figsize=(10,7))
plt.plot(x_range, history_x, 'o--',color='dodgerblue', label='x')
plt.plot(x_range, history_x_prime,'D--', color='orange', label='x\'')

plt.title("Fig. 3 Polynomial Convergent Solution")
```

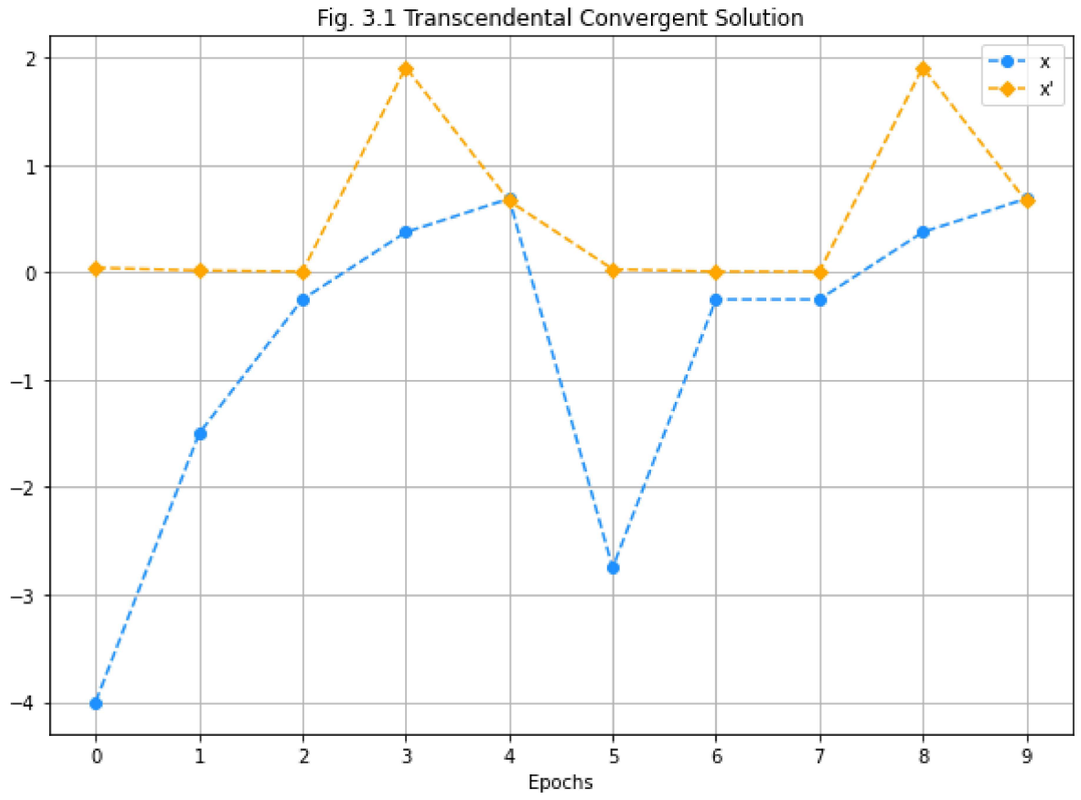
```
plt.title( Fig. 3 Polynomial Convergent Solution )
plt.xlabel("Epochs")
plt.xticks(np.arange(min(x_range), max(x_range)+1, 1.0))
plt.legend()
plt.grid()
plt.show()
```



It can be seen in the graph that in 4 epochs, the x value (dodger blue) and the computed value (orange) meets. This means that the polynomial convergent solution using the developed bisection method was excellent since the values met at some point.

```
def bisection(func,i1,i2,n_roots,epochs,n_move,tol = 1.0e-06):
  fpoint = []
  spoint = []
  history_x =[]
  history_x_prime = []

  for i in range(n_move): #Moving intervals
    i1+=0.25
    i2+=0.25
    fpoint.append(i1)
    spoint.append(i2)

  for (i1,i2) in zip (fpoint,spoint):
    y1, y2 = func(i1), func(i2)
    root = None
    end_bisect = 0
    if np.sign(y1) == np.sign(y2):
      pass #Root are not in this interval
    else:
      for bisect in range(epochs):
        midp = np.mean([i1,i2])
        y_mid = func(midp)
        y1 = func(i1)
        history_x.append(i1)
        history_x_prime.append(y1)
        if np.allclose(0,y1, tol):
          root = i1
          break
        if np.sign(y1) != np.sign(y_mid): #root is in first-half interval
          i2 = midp
```

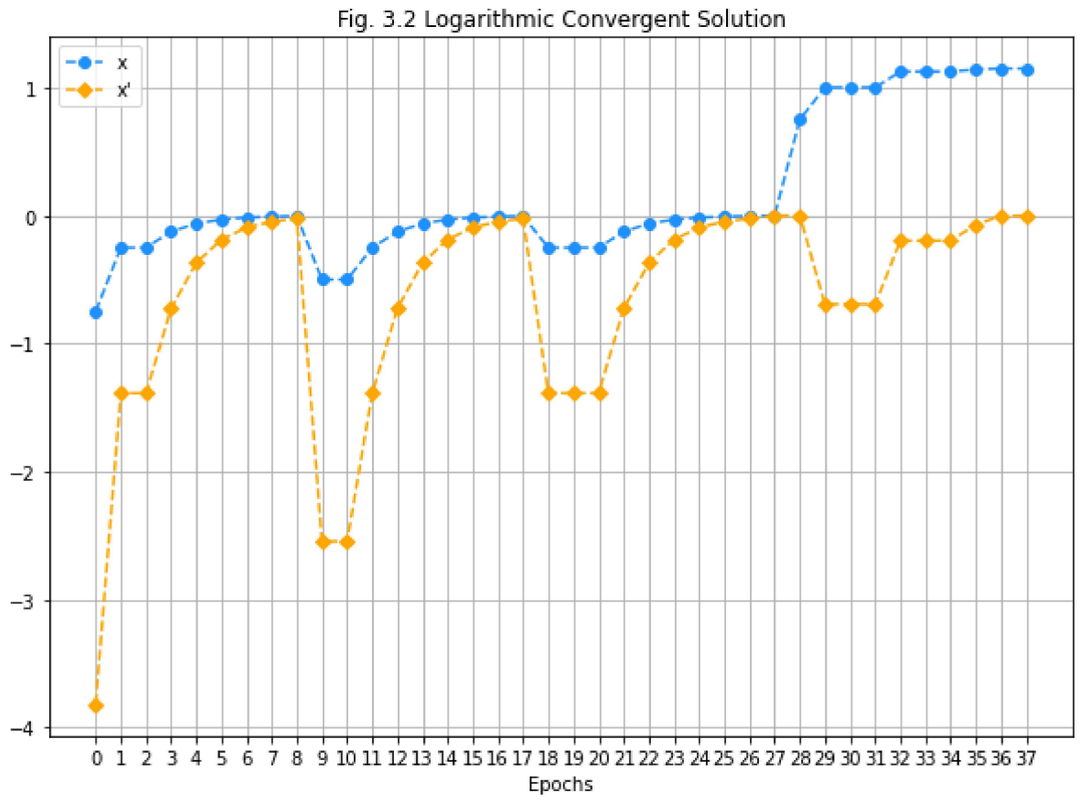
```
        else: #root is in second-half interval
          i1 = midp

  return history_x,history_x_prime

func = lambda x: np.sin(5*x)-np.cos(2**4)

history_x,history_x_prime = bisection(func,i1 = -5,i2 = 0,n_roots = 5,epochs = 5,n_move =10,t

history_x = history_x[0:]
history_x_prime = history_x_prime[0:]
x_range = np.arange(len(history_x),dtype=int)

plt.figure(figsize=(10,7))
plt.plot(x_range, history_x, 'o--',color='dodgerblue', label='x')
plt.plot(x_range, history_x_prime,'D--', color='orange', label='x\'')

plt.title("Fig. 3.1 Transcendental Convergent Solution")
plt.xlabel("Epochs")
plt.xticks(np.arange(min(x_range), max(x_range)+1, 1.0))
plt.legend()
plt.grid()
plt.show()
```



It can be seen in the graph that in 4 epochs, the x value (dodger blue) and the computed value (orange) meets. This means that the transcendental convergent solution using the developed bisection method was excellent since the values met at some point.

```
def bisection(func,i1,i2,n_roots,epochs,n_move,tol = 1.0e-06):
  fpoint = []
  spoint = []
  history_x =[]
  history_x_prime = []

  for i in range(n_move): #Moving intervals
    i1+=0.25
    i2+=0.25
    fpoint.append(i1)
    spoint.append(i2)
```

```
  for (i1,i2) in zip (fpoint,spoint):
    y1, y2 = func(i1), func(i2)
    root = None
    end_bisect = 0
    if np.sign(y1) == np.sign(y2):
      pass #Root are not in this interval
    else:
      for bisect in range(epochs):
        midp = np.mean([i1,i2])
        y_mid = func(midp)
        y1 = func(i1)
        history_x.append(i1)
        history_x_prime.append(y1)
        if np.allclose(0,y1, tol):
          root = i1
          break
        if np.sign(y1) != np.sign(y_mid): #root is in first-half interval
          i2 = midp
        else: #root is in second-half interval
          i1 = midp

  return history_x,history_x_prime

func = lambda x: np.log(4*x**2-3*x+1)*(x**5-2)

history_x,history_x_prime = bisection(func,i1 = -1,i2 = 0,n_roots = 5,epochs = 9,n_move =10,t

history_x = history_x[0:]
history_x_prime = history_x_prime[0:]
x_range = np.arange(len(history_x),dtype=int)

plt.figure(figsize=(10,7))
plt.plot(x_range, history_x, 'o--',color='dodgerblue', label='x')
plt.plot(x_range, history_x_prime,'D--', color='orange', label='x\'')

plt.title("Fig. 3.2 Logarithmic Convergent Solution")
plt.xlabel("Epochs")
plt.xticks(np.arange(min(x_range), max(x_range)+1, 1.0))
plt.legend()
plt.grid()
plt.show()
```

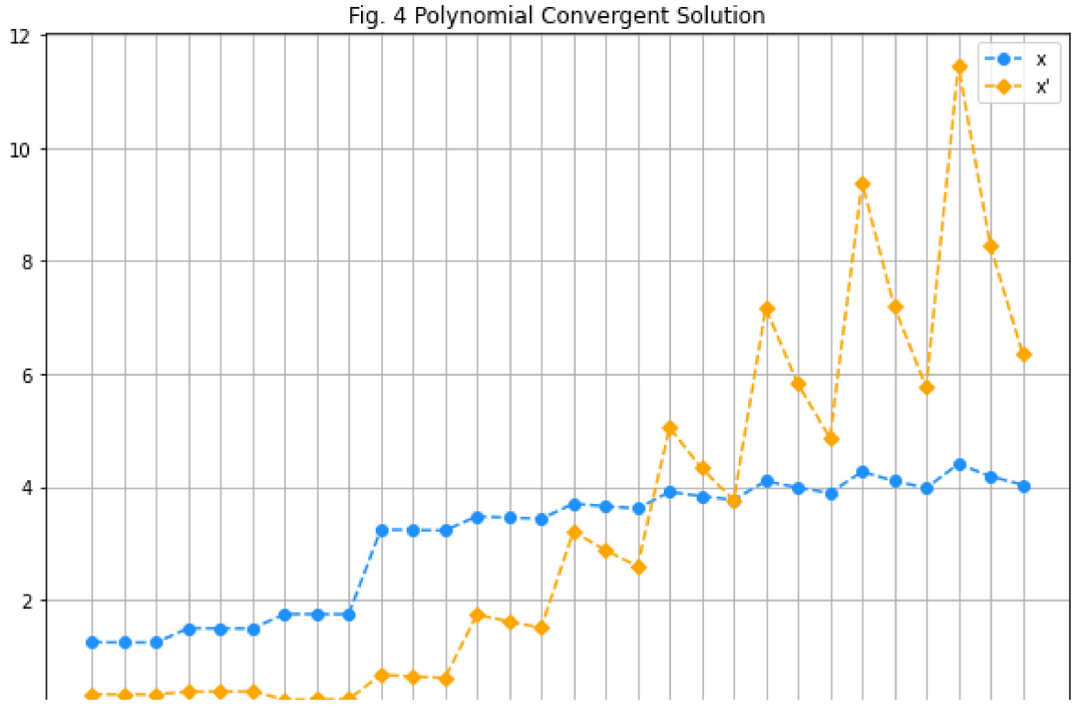


It can be seen in the graph that in 8 epochs, the x value (dodger blue) and the computed value (orange) meets. This means that the logarithmic convergent solution using the developed bisection method was excellent since the values met at some point.

Falsi Method

```
def falsi(func,a,b,n_roots,epochs,n_move,tol = 1.0e-06):
  x_roots = []
  fpoint = []
  spoint = []
  history_x =[]
  history_x_prime = []


  for i in range(n_move):
    a+=0.25
    b+=0.25
    fpoint.append(a)
    spoint.append(b)

  for (a,b) in zip (fpoint,spoint):
    y1, y2 = func(a), func(b)
    root = None
    pos = 0
    if np.allclose(0,y1): root = a
    elif np.allclose(0,y2): root = b
    elif np.sign(y1) == np.sign(y2):
      pass
      #print("No root here")
    else:
      for pos in range(epochs):
        c = b - (func(b)*(b-a))/(func(b)-func(a)) ##false root
        history_x.append(c)
        history_x_prime.append(func(c))
        if np.allclose(0,func(c), tol):
          root = c
          break
        if np.sign(func(a)) != np.sign(func(c)): b,y2 = c,func(c)
        else: a,y1 = c,func(c)
  return history_x,history_x_prime

func = lambda x: x**3-6*x**2+11*x-6
history_x,history_x_prime= falsi(func,a = -10,b = 1,n_roots = 1,epochs = 3,n_move = 15,tol =

history_x = history_x[0:]
history_x_prime = history_x_prime[0:]
x_range = np.arange(len(history_x),dtype=int)

plt.figure(figsize=(10,7))
plt.plot(x_range, history_x, 'o--',color='dodgerblue', label='x')
plt.plot(x_range, history_x_prime,'D--', color='orange', label='x\'')

plt.title("Fig. 4 Polynomial Convergent Solution")
plt.xlabel("Epochs")
plt.xticks(np.arange(min(x_range), max(x_range)+1, 1.0))
plt.legend()
plt.grid()
plt.show()
```

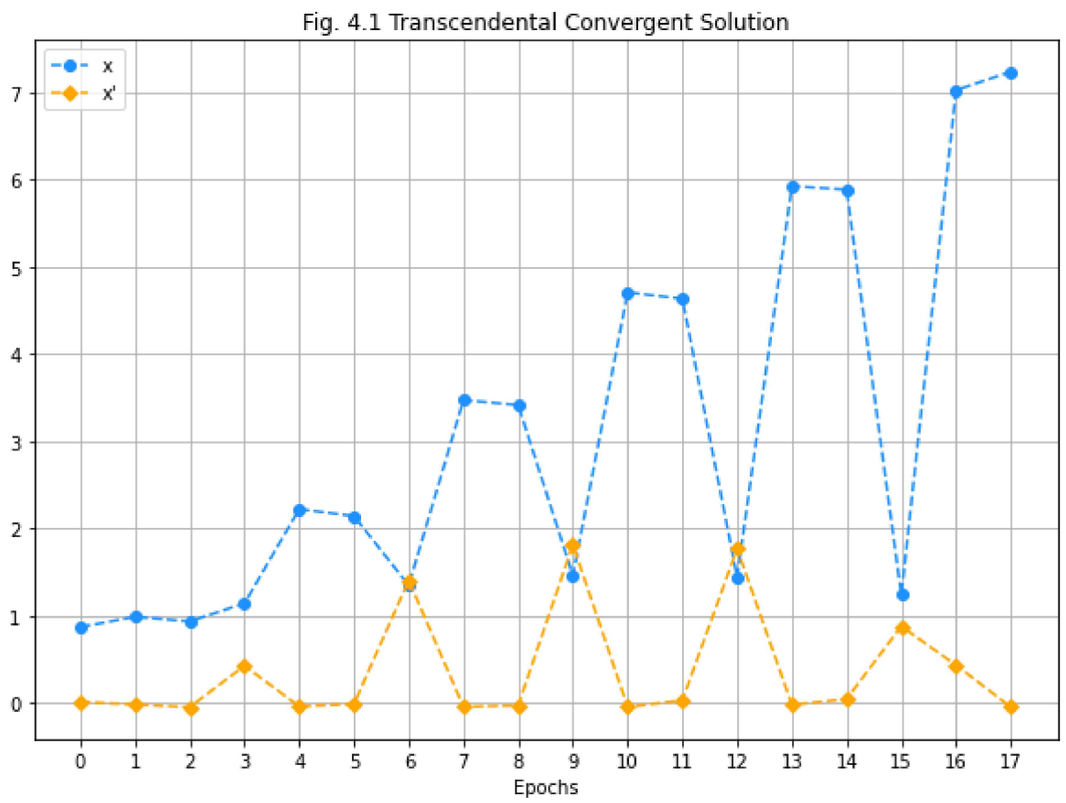
Fig. 4 Polynomial Convergent Solution



It can be seen in the graph that in 20 epochs, the x value (dodger blue) and the computed value (orange) meets. This means that the transcendental convergent solution using the developed falsi method was good since the values met at some point. It can be observed in the graph that the x value and computed value diverge as more epochs go by and this may be due to the intervals used in the method or the algorithm created was unoptimized. However, the x value and computed value did meet so the solution succesfully converged.

```
def falsi(func,a,b,n_roots,epochs,n_move,tol = 1.0e-06):
  x_roots = []
  fpoint = []
  spoint = []
  history_x =[]
  history_x_prime = []


  for i in range(n_move):
    a+=0.25
    b+=0.25
    fpoint.append(a)
    spoint.append(b)

  for (a,b) in zip (fpoint,spoint):
    y1, y2 = func(a), func(b)
    root = None
    pos = 0
    if np.allclose(0,y1): root = a
    elif np.allclose(0,y2): root = b
    elif np.sign(y1) == np.sign(y2):
      pass
      #print("No root here")
    else:
      for pos in range(epochs):
        c = b - (func(b)*(b-a))/(func(b)-func(a)) ##false root
        history_x.append(c)
        history_x_prime.append(func(c))
        if np.allclose(0,func(c), tol):
          root = c
          break
        if np.sign(func(a)) != np.sign(func(c)): b,y2 = c,func(c)
        else: a,y1 = c,func(c)
  return history_x,history_x_prime

func = lambda x: np.sin(5*x)-np.cos(2**4)
```

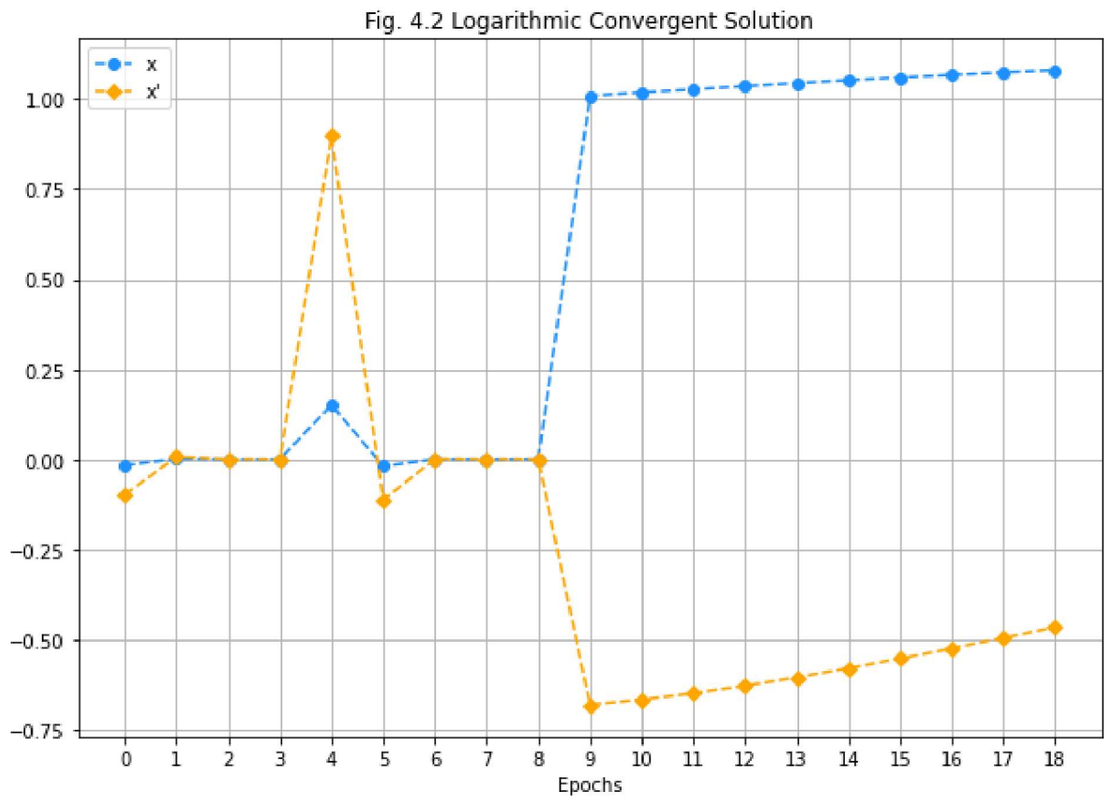
```
history_x,history_x_prime= falsi(func,a = -10,b = -2.5,n_roots = 1,epochs = 3,n_move = 40,tol

history_x = history_x[0:]
history_x_prime = history_x_prime[0:]
x_range = np.arange(len(history_x),dtype=int)

plt.figure(figsize=(10,7))
plt.plot(x_range, history_x, 'o--',color='dodgerblue', label='x')
plt.plot(x_range, history_x_prime,'D--', color='orange', label='x\'')

plt.title("Fig. 4.1 Transcendental Convergent Solution")
plt.xlabel("Epochs")
plt.xticks(np.arange(min(x_range), max(x_range)+1, 1.0))
plt.legend()
plt.grid()
plt.show()
```



It can be seen in the graph that in 6 epochs, the x value (dodger blue) and the computed value (orange) meets. This means that the logarithmic convergent solution using the developed falsi method was good since the values met at some point. It can be observed in the graph that the x value and computed value diverge as more epochs go by and this may be due to the intervals used in the method or the algorithm created was unoptimized. However, the x value and computed value did meet so the solution succesfully converged.

```
def falsi(func,a,b,n_roots,epochs,n_move,tol = 1.0e-06):
  x_roots = []
  fpoint = []
  spoint = []
  history_x =[]
  history_x_prime = []


  for i in range(n_move):
    a+=0.25
    b+=0.25
    fpoint.append(a)
    spoint.append(b)

  for (a,b) in zip (fpoint,spoint):
    y1, y2 = func(a), func(b)
```

```
    y1, y2 = func(a), func(b)
    root = None
    pos = 0
    if np.allclose(0,y1): root = a
    elif np.allclose(0,y2): root = b
    elif np.sign(y1) == np.sign(y2):
      pass
      #print("No root here")
    else:
      for pos in range(epochs):
        c = b - (func(b)*(b-a))/(func(b)-func(a)) ##false root
        history_x.append(c)
        history_x_prime.append(func(c))
        if np.allclose(0,func(c), tol):
          root = c
          break
        if np.sign(func(a)) != np.sign(func(c)): b,y2 = c,func(c)
        else: a,y1 = c,func(c)
  return history_x,history_x_prime

func = lambda x: np.log(4*x**2-3*x+1)*(x**5-2)
history_x,history_x_prime= falsi(func,a = -1,b = 0,n_roots = 10,epochs = 10,n_move = 20,tol =

history_x = history_x[0:]
history_x_prime = history_x_prime[0:]
x_range = np.arange(len(history_x),dtype=int)

plt.figure(figsize=(10,7))
plt.plot(x_range, history_x, 'o--',color='dodgerblue', label='x')
plt.plot(x_range, history_x_prime,'D--', color='orange', label='x\'')

plt.title("Fig. 4.2 Logarithmic Convergent Solution")
plt.xlabel("Epochs")
plt.xticks(np.arange(min(x_range), max(x_range)+1, 1.0))
plt.legend()
plt.grid()
plt.show()
```



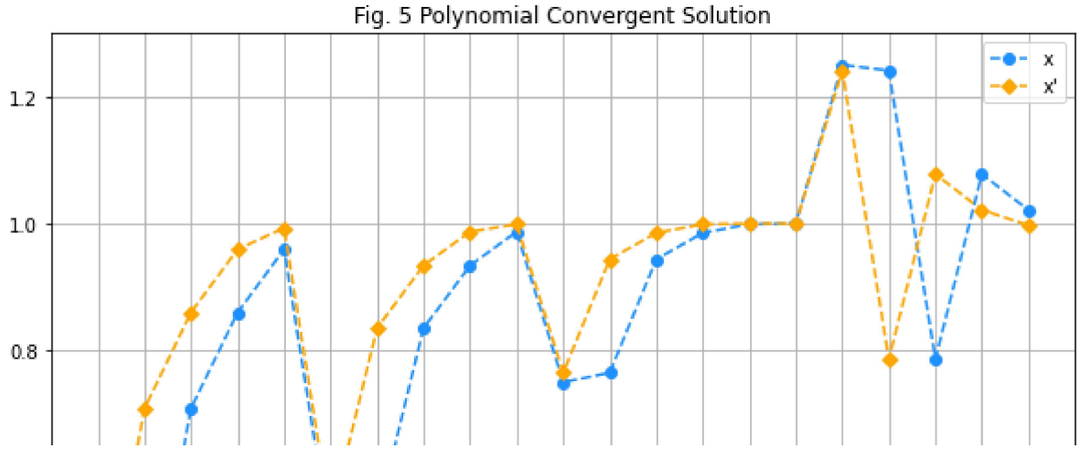
It can be seen in the graph that in 1 epochs, the x value (dodger blue) and the computed value (orange) meets. This means that the logarithmic convergent solution using the developed falsi method was good since the values met at some point. It can be observed in the graph that the x

value and computed value diverge as more epochs go by and this may be due to the intervals used in the method or the algorithm created was unoptimized. However, the x value and computed value did meet so the solution succesfully converged.

Secant Method

```
def secant(func,a,b,n_roots,epochs,n_move,tol = 1.0e-06):
  fpoint = []
  spoint = []
  history_x = []
  history_x_prime = []

  for i in range(n_move):
    a+=0.25
    fpoint.append(a)
    b+=0.25
    spoint.append(b)
    #print(fpoint)
    #print(spoint)

  for (a,b) in zip (fpoint,spoint):
    root = None
    end_epoch =  0
    for epoch in range(epochs):
      c = b - (func(b)*(b-a))/(func(b)-func(a))
      history_x.append(b)
      history_x_prime.append(c)
      if np.allclose(b,c):
        root = c
        break
      else:
        a,b = b,c

  return history_x, history_x_prime

func = lambda x: x**3-6*x**2+11*x-6
history_x,history_x_prime = secant(func,a = -5,b = 0,n_roots = 4,epochs = 5,n_move = 5,tol =

history_x = history_x[0:]
history_x_prime = history_x_prime[0:]
x_range = np.arange(len(history_x),dtype=int)

plt.figure(figsize=(10,7))
plt.plot(x_range, history_x, 'o--',color='dodgerblue', label='x')
plt.plot(x_range, history_x_prime,'D--', color='orange', label='x\'')

plt.title("Fig. 5 Polynomial Convergent Solution")
plt.xlabel("Epochs")
plt.xticks(np.arange(min(x_range), max(x_range)+1, 1.0))
plt.legend()
plt.grid()
plt.show()
```

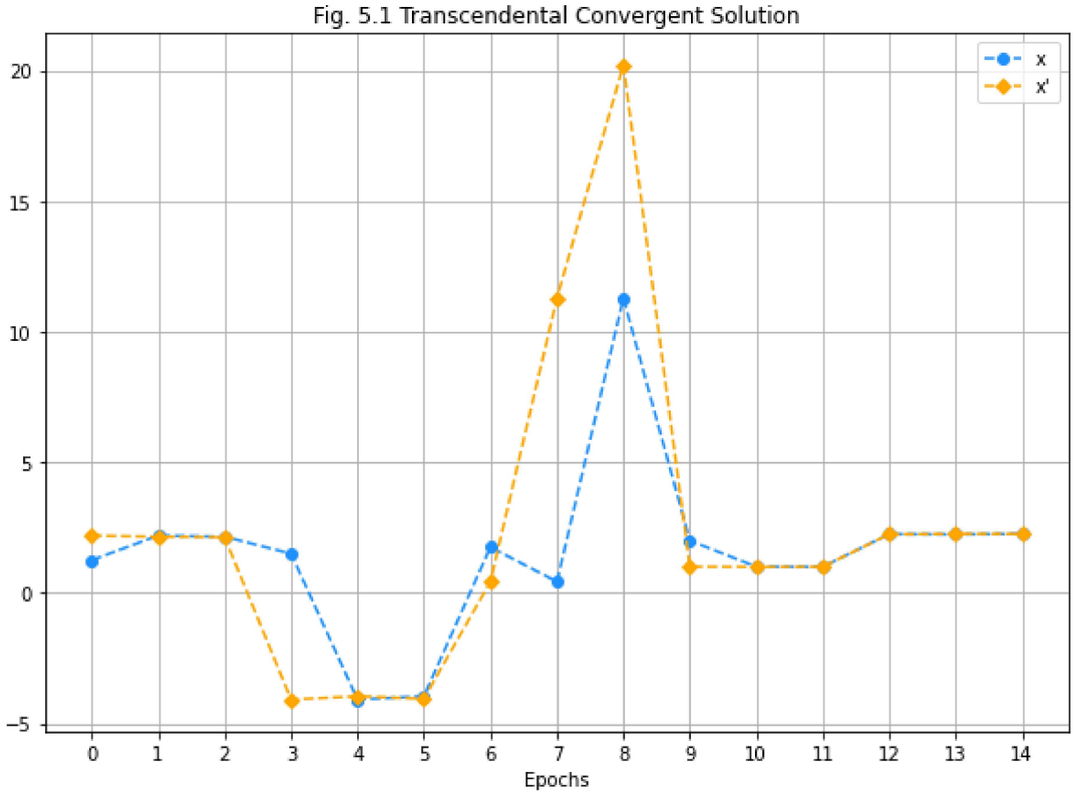
Fig. 5 Polynomial Convergent Solution



It can be seen in the graph that in 14 epochs, the x value (dodger blue) and the computed value (orange) meets. This means that the polynomial convergent solution using the developed secant method was excellent since the values met at some point.

```
def secant(func,a,b,n_roots,epochs,n_move,tol = 1.0e-06):
  fpoint = []
  spoint = []
  history_x = []
  history_x_prime = []

  for i in range(n_move):
    a+=0.25
    fpoint.append(a)
    b+=0.25
    spoint.append(b)
    #print(fpoint)
    #print(spoint)

  for (a,b) in zip (fpoint,spoint):
    root = None
    end_epoch =  0
    for epoch in range(epochs):
      c = b - (func(b)*(b-a))/(func(b)-func(a))
      history_x.append(b)
      history_x_prime.append(c)
      if np.allclose(b,c):
        root = c
        break
      else:
        a,b = b,c

  return history_x, history_x_prime

func = lambda x: np.sin(5*x)-np.cos(2**4)
history_x,history_x_prime = secant(func,a = 0,b = 1,n_roots = 4,epochs = 3,n_move = 5,tol = 1

history_x = history_x[0:]
history_x_prime = history_x_prime[0:]
x_range = np.arange(len(history_x),dtype=int)

plt.figure(figsize=(10,7))
plt.plot(x_range, history_x, 'o--',color='dodgerblue', label='x')
plt.plot(x_range, history_x_prime,'D--', color='orange', label='x\'')

plt.title("Fig. 5.1 Transcendental Convergent Solution")
plt.xlabel("Epochs")
plt.xticks(np.arange(min(x_range), max(x_range)+1, 1.0))
plt.legend()
plt.grid()
plt.show()
```

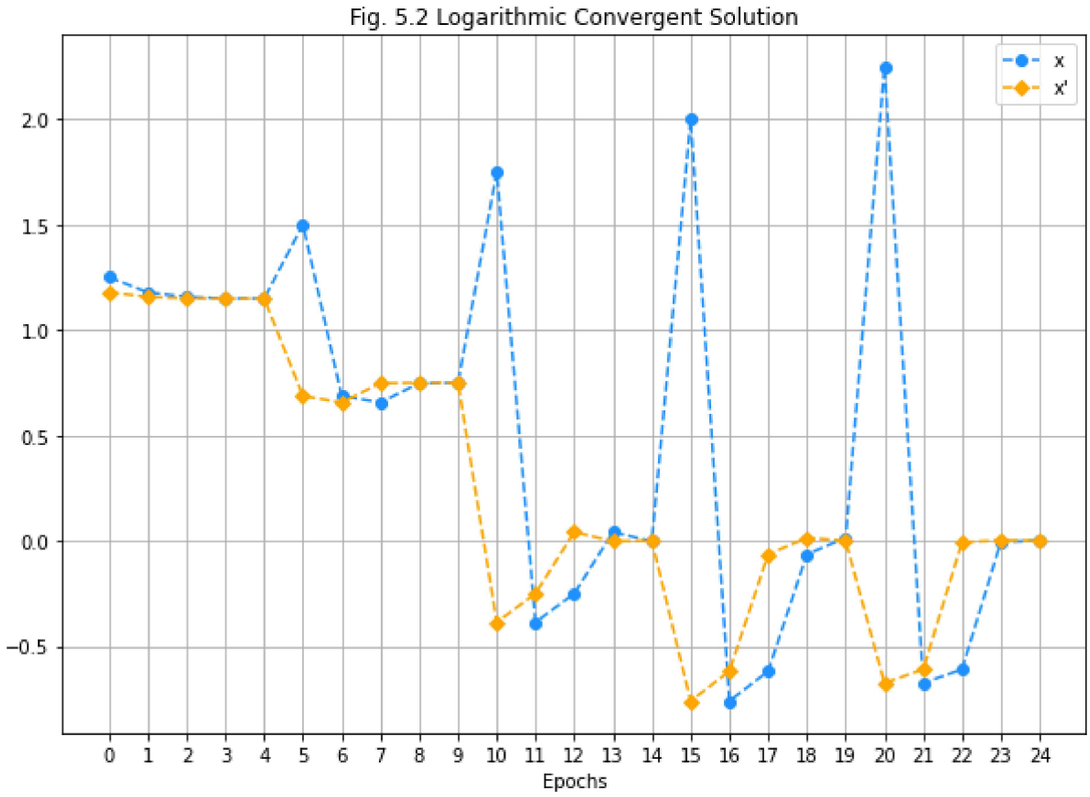
Fig. 5.1 Transcendental Convergent Solution



It can be seen in the graph that in just 1 epoch, the x value (dodger blue) and the computed value (orange) meets. This means that the transcendental convergent solution using the developed secant method was excellent since the values met at some point.

```
def secant(func,a,b,n_roots,epochs,n_move,tol = 1.0e-06):
  fpoint = []
  spoint = []
  history_x = []
  history_x_prime = []

  for i in range(n_move):
    a+=0.25
    b+=0.25
    fpoint.append(a)
    spoint.append(b)

  for (a,b) in zip (fpoint,spoint):
    root = None
    end_epoch =  0
    for epoch in range(epochs):
      c = b - (func(b)*(b-a))/(func(b)-func(a))
      history_x.append(b)
      history_x_prime.append(c)
      if np.allclose(b,c):
        root = c
        break
      else:
        a,b = b,c

  return history_x, history_x_prime

func = lambda x: np.log(4*x**2-3*x+1)*(x**5-2)
history_x,history_x_prime = secant(func,a = -2,b = 1,n_roots = 4,epochs = 5,n_move = 5,tol =

history_x = history_x[0:]
history_x_prime = history_x_prime[0:]
x_range = np.arange(len(history_x),dtype=int)

plt.figure(figsize=(10,7))
plt.plot(x_range, history_x, 'o--',color='dodgerblue', label='x')
plt.plot(x_range, history_x_prime,'D--', color='orange', label='x\'')
```

```
plt.plot(x_range, history_x_prime, c='orange', label="x'")

plt.title("Fig. 5.2 Logarithmic Convergent Solution")
plt.xlabel("Epochs")
plt.xticks(np.arange(min(x_range), max(x_range)+1, 1.0))
plt.legend()
plt.grid()
plt.show()
```



It can be seen in the graph that in 2 epochs, the x value (dodger blue) and the computed value (orange) meets. This means that the logarithmic convergent solution using the developed secant method was excellent since the values met at some point.

Conclusion

Based on the result of the graphs, it is to be concluded that the fastest algorithm to use in finding the roots of a polynomial function is the bisection method. For the roots of a transcendental function, the fastest method to use is the brute force method while for the roots of a logarithmic function, the fastest is the falsi method. However, in terms of the consistency in the convergence of the solutions, the best method in every function was the brute force method since the graph all had a clean converging pattern.

The comparison made are purely based on the test case scenario in this notebook and considering the inconsistent initial guesses for intervals is one of the factor that affects the result of the ranking of covergence rate of each method.