Implementações de Cálculo Numérico

Amanda Lopes Dantas 13/0100391
Caroline Mansur Araujo e Silva 16/0116104
Emily da Costa Santos 13/0025348
Taian Cristal Ferreira Salles 11/0140826

Lista 1

```
# -*- coding: latin-1 -*-

"""
* Resolução do exercício 1 do capítulo 1.1 (Timothy Sauer. Numerical Analysis. Pearson, 2ª
Edição)
*
* Executado como : python bisseccao_1.1-1.py <letra> <a> <b> <numero_de_iterações>
*		<letra> -> qual as letras o exercício (a, b ou c)
*		<a> e <b> -> representam o intervalo [a,b]
*
* Parâmetros usados para teste:
*			python bisseccao_1.1-1.py a 2 3 20
*			python bisseccao_1.1-1.py b 1 2 20
*			python bisseccao_1.1-1.py c 6 7 20
*
"""

import sys
import math

def fa(x):
	return x**3 - 9 #essa eh a funcao (a)

def fb(x):
	return 3*x**3 + x**2 - x - 5 #essa eh a funcao (b)

def fc(x):
	return (math.cos(x) * math.cos(x)) + 6 -x #essa eh a funcao (c)

# a tolerancia eh o numero de casas finais
def bisseccao (letra,a,b,n):

	if(letra == 'a'):
		def f(x):
			return fa(x)
	elif (letra == 'b'):
		def f(x):
			return fb(x)
	else:
		def f(x):
			return fc(x)

	if f(a) * f(b) >= 0:
		print ('Erro! f(a)f(b) < 0 não ocorre')
		return None
	else:
		i = 1
		#while ((b-a)/2.0 > tolerancia):
		while (i <= n):
```

```
            c = (a+b)/2.0
            if f(c) == 0:
                return c
            elif f(a)*f(c) < 0:
                b = c
                i = i +1
            else:
                a = c
                i = i +1


        return c


def main (argv) :

    if(len(sys.argv) != 5):
        sys.exit("Parâmetros errados: bisecao_1.1-1.py <letra> <a> <b> <numero de
iterações>")

    if(sys.argv[1] not in ['a','b','c']):
        sys.exit("Parâmetro de letra errado! São apenas válidos a, b ou c.")


    res = bisseccao (sys.argv[1], int(sys.argv[2]), int (sys.argv[3]), int(sys.argv[4]))
    if (res != None):
        print ('\n\tA raiz da letra é: '),
        print res,
        print ('\n')


if __name__== "__main__":
    main(sys.argv[1:])
```

```
# -*- coding: latin-1 -*-

"""
* Resolução do exercício 2 do capítulo 1.1 (Timothy Sauer. Numerical Analysis. Pearson, 2ª
Edição)
*
* Executado como : python bisseccao_1.1-2.py <letra> <a> <b> <numero_de_iterações>
*           <letra> -> qual as letras o exercício (a, b ou c)
*           <a> e <b> -> representam o intervalo [a,b]
*
* Parâmetros usados para teste:
*                   python bisseccao_1.1-2.py a 0 1 27
*                   python bisseccao_1.1-2.py b -1 0 27
*                   python bisseccao_1.1-2.py c 1 2 27
*
"""
```

```
import sys
import math

def fa(x):
	return x**5 +x - 1 #essa eh a funcao (a)

def fb(x):
	return math.sin(x) - 6*x - 5 #essa eh a funcao (b)

def fc(x):
	return math.log(x) + x**2 - 3 #essa eh a funcao (c)

# a tolerancia eh o numero de casas finais
def bisseccao (letra,a,b,n):

	if(letra == 'a'):
		def f(x):/
			return fa(x)
	elif (letra == 'b'):
		def f(x):
			return fb(x)
	else:
		def f(x):
			return fc(x)

	if f(a) * f(b) >= 0:
		print ('Erro! f(a)f(b) < 0 não ocorre')
		return None
	else:
		i = 1
		#while ((b-a)/2.0 > tolerancia):
		while (i <= n):
			c = (a+b)/2.0
			if f(c) == 0:
				return c
			elif f(a)*f(c) < 0:
				b = c
				i = i +1
			else:
				a = c
				i = i +1


		return c


def main (argv) :

	if(len(sys.argv) != 5):
		sys.exit("Parâmetros errados: bisecao_1.1-2.py <letra> <a> <b> <numero de
iterações>")
```

```
	if(sys.argv[1] not in ['a','b','c']):
		sys.exit("Parâmetro de letra errado! São apenas válidos a, b ou c.")


	res = bisseccao (sys.argv[1], int(sys.argv[2]), int (sys.argv[3]), int(sys.argv[4]))
	if (res != None):
		print ('\n\tA raiz da letra é: '),
		print res,
		print ('\n')


if __name__== "__main__":
	main(sys.argv[1:])
```

```
# -*- coding: latin-1 -*-

"""
* Resolução do exercício 4 do capítulo 1.1 (Timothy Sauer. Numerical Analysis. Pearson, 2ª Edição)
*
* Executado como : python bisseccao_1.1-4.py <letra> <a> <b> <numero_de_iterações>
*		<letra> -> qual as letras o exercício (a, b ou c)
*		<a> e <b> -> representam o intervalo [a,b]
*
* Parâmetros usados para teste:
*			python bisseccao_1.1-4.py a 1 2 27
*			python bisseccao_1.1-4.py b 1 2 27
*			python bisseccao_1.1-4.py c 2 3 27
*
"""

import sys
import math

def fa(x):
	return x**2 - 2 #essa eh a funcao (a)

def fb(x):
	return x**2 - 3 #essa eh a funcao (b)

def fc(x):
	return x**2 - 5 #essa eh a funcao (c)

# a tolerancia eh o numero de casas finais
def bisseccao (letra,a,b,n):

	if(letra == 'a'):
		def f(x):
			return fa(x)
	elif (letra == 'b'):
		def f(x):
```

```
            return fb(x)
    else:
        def f(x):
            return fc(x)

    if f(a) * f(b) >= 0:
        print ('Erro! f(a)f(b) < 0 não ocorre')
        return None
    else:
        i = 1
        #while ((b-a)/2.0 > tolerancia):
        while (i <= n):
            c = (a+b)/2.0
            if f(c) == 0:
                return c
            elif f(a)*f(c) < 0:
                b = c
                i = i +1
            else:
                a = c
                i = i +1


        return c


def main (argv) :

    if(len(sys.argv) != 5):
        sys.exit("Parâmetros errados: bisecao_1.1-2.py <letra> <a> <b> <numero de
iterações>")

    if(sys.argv[1] not in ['a','b','c']):
        sys.exit("Parâmetro de letra errado! São apenas válidos a, b ou c.")


    res = bisseccao (sys.argv[1], int(sys.argv[2]), int (sys.argv[3]), int(sys.argv[4]))
    if (res != None):
        print ('\n\tA raiz da letra é: '),
        print res,
        print ('\n')


if __name__== "__main__":
    main(sys.argv[1:])
```

```
# -*- coding: latin-1 -*-

"""
* Resolução do exercício 5 do capítulo 1.1 (Timothy Sauer. Numerical Analysis. Pearson, 2ª
Edição)
```

```
*
* Executado como : python bisseccao_1.1-5.py <letra> <a> <b> <numero_de_iterações>
*               <letra> -> qual as letras o exercício (a, b ou c)
*               <a> e <b> -> representam o intervalo [a,b]
*
* Parâmetros usados para teste:
*                          python bisseccao_1.1-5.py a 1 2 27
*                          python bisseccao_1.1-5.py b 1 2 27
*                          python bisseccao_1.1-5.py c 1 2 27
*
"""

import sys
import math

def fa(x):
	return x**3 - 2 #essa eh a funcao (a)

def fb(x):
	return x**3 - 3 #essa eh a funcao (b)

def fc(x):
	return x**3 - 5 #essa eh a funcao (c)

# a tolerancia eh o numero de casas finais
def bisseccao (letra,a,b,n):

	if(letra == 'a'):
		def f(x):
			return fa(x)
	elif (letra == 'b'):
		def f(x):
			return fb(x)
	else:
		def f(x):
			return fc(x)

	if f(a) * f(b) >= 0:
		print ('Erro! f(a)f(b) < 0 não ocorre')
		return None
	else:
		i = 1
		#while ((b-a)/2.0 > tolerancia):
		while (i <= n):
			c = (a+b)/2.0
			if f(c) == 0:
				return c
			elif f(a)*f(c) < 0:
				b = c
				i = i +1
			else:
				a = c
```

```
                i = i +1


        return c


def main (argv) :

    if(len(sys.argv) != 5):
        sys.exit("Parâmetros errados: bisecao_1.1-2.py <letra> <a> <b> <numero de iterações>")

    if(sys.argv[1] not in ['a','b','c']):
        sys.exit("Parâmetro de letra errado! São apenas válidos a, b ou c.")


    res = bisseccao (sys.argv[1], int(sys.argv[2]), int (sys.argv[3]), int(sys.argv[4]))
    if (res != None):
        print ('\n\tA raiz da letra é: '),
        print res,
        print ('\n')


if __name__== "__main__":
    main(sys.argv[1:])
```

Lista 2

```
# -*- coding: latin-1 -*-
"""
* Resolução do exercício 1 do capítulo 1.2 (Timothy Sauer. Numerical Analysis. Pearson, 2ª Edição)
*
* Executado como : python ponto-fixo_1.2-1.py <letra> <x>
*           <letra> -> qual as letras o exercício (a, b ou c)
*           <x> -> representam o chute inicial xo
*
* Parâmetros usados para teste:
*                   python ponto-fixo_1.2-1.py a 1
*                   python ponto-fixo_1.2-1.py b 1
*                   python ponto-fixo_1.2-1.py c 1
*
"""
import math
import sys


def ga(x):
    return (2*x + 2) **(1/3.0) #essa é a função (a)
def gb(x):
    return math.log(7-x) #essa é a função (b)
```

```
def gc(x):
	return math.log(4 - math.sin(x)) #essa é a função (c)

MAX = 200 #constante do número máximo de iterações

def ponto_fixo(letra,x):

	if (letra == 'a'):
		def g(x):
			return ga(x)
	elif (letra == 'b'):
		def g(x):
			return gb(x)
	else:
		def g(x):
			return gc(x)

	i = 0
	erro = 1
	x_novo = x

	while ((erro >.5E-8) and (i < MAX)):
		print '\t\t',i,'\t', x_novo,'\t', g(x_novo)
		x_antigo = x_novo
		x_novo = g(x_novo)
		i = i +1
		erro = abs(x_novo - x_antigo)/ (abs(x_novo) + 0.000000001)

	if (i == 200):
		print 'Limite de 200 iterações atingido'
	else:
		print 'A raiz é: ',
		print ("%.8f" % x_novo)

def main (argv):

	if (len(sys.argv) != 3):
		sys.exit('Parâmetros errados: ponto_fixo.py <letra> <x>')

	print '\tTABELA:\n\n\t\ti\txi\tg(xi)\n'
	res = ponto_fixo (sys.argv[1], float (sys.argv[2]))


if __name__ == '__main__':
	main(sys.argv[1:])
```

```
# -*- coding: latin-1 -*-
"""
* Resolução do exercício 2 do capítulo 1.2 (Timothy Sauer. Numerical Analysis. Pearson, 2ª Edição)
*
```

```
* Executado como : python ponto-fixo_1.2-2.py <letra> <x>
*               <letra> -> qual as letras o exercício (a, b ou c)
*               <x> -> representam o chute inicial xo
*
* Parâmetros usados para teste:
*                       python ponto-fixo_1.2-2.py a 1
*                       python ponto-fixo_1.2-2.py b 0
*                       python ponto-fixo_1.2-2.py c 1
*
"""
import math
import sys


def ga(x):
	return (1 + 2 * x ** 5.0) / (1 + 3*x**4.0) #essa é a função (a)
def gb(x):
	return (math.sin(x) - 5)/6.0 #essa é a função (b)
def gc(x):
	return (3 - math.log(x)) ** (1/2.0) #essa é a função (c)

MAX = 200 #constante do número máximo de iterações

def ponto_fixo(letra,x):

	if (letra == 'a'):
		def g(x):
			return ga(x)
	elif (letra == 'b'):
		def g(x):
			return gb(x)
	else:
		def g(x):
			return gc(x)

	i = 0
	erro = 1
	x_novo = x

	while ((erro >.5E-8) and (i < MAX)):
		print '\t\t',i,'\t', x_novo,'\t', g(x_novo)
		x_antigo = x_novo
		x_novo = g(x_novo)
		i = i +1
		erro = abs(x_novo - x_antigo)/ (abs(x_novo) + 0.000000001)

	if (i == 200):
		print 'Limite de 200 iterações atingido'
	else:
		print 'A raiz é: ',
		print ("%.8f" % x_novo)
```

```
def main (argv):

	if (len(sys.argv) != 3):
		sys.exit('Parâmetros errados: ponto_fixo.py <letra> <x>')

	print '\tTABELA:\n\n\t\ti\txi\tg(xi)\n'
	res = ponto_fixo (sys.argv[1], float (sys.argv[2]))


if __name__ == '__main__':
	main(sys.argv[1:])
```

```
# -*- coding: latin-1 -*-
"""
* Resolução do exercício 4 do capítulo 1.2 (Timothy Sauer. Numerical Analysis. Pearson, 2ª
Edição)
*
* Executado como : python ponto-fixo_1.2-2.py <letra> <x>
*		<letra> -> qual as letras o exercício (a, b ou c)
*		<x> -> representam o chute inicial xo
*
* Parâmetros usados para teste:
*			python bisseccao_1.2-4.py a 1
*			python bisseccao_1.2-2.py b 1
*			python bisseccao_1.2-2.py c 1
*
"""
import math
import sys


def ga(x):
	return (2*x + 2/ x ** 2) / 3.0 #essa é a função (a)
def gb(x):
	return (2*x + 3 / x ** 2) / 3.0 #essa é a função (b)
def gc(x):
	return (2*x + 5 / x ** 2) / 3.0 #essa é a função (c)

MAX = 200 #constante do número máximo de iterações

def ponto_fixo(letra,x):

	if (letra == 'a'):
		def g(x):
			return ga(x)
	elif (letra == 'b'):
		def g(x):
			return gb(x)
	else:
		def g(x):
			return gc(x)
```

```
	i = 0
	erro = 1
	x_novo = x

	while ((erro >.5E-8) and (i < MAX)):
		print '\t\t',i,'\t', x_novo,'\t', g(x_novo)
		x_antigo = x_novo
		x_novo = g(x_novo)
		i = i +1
		erro = abs(x_novo - x_antigo)/ (abs(x_novo) + 0.000000001)

	if (i == 200):
		print 'Limite de 200 iterações atingido'
	else:
		print '\nA raiz é: ',
		print ("%.8f" % x_novo)
		print 'O número de iterações necessárias foi: ',
		print ("%d" % i)
		print 'Chute inicial: ',
		print ("%d\n" % x)

def main (argv):

	if (len(sys.argv) != 3):
		sys.exit('Parâmetros errados: ponto_fixo.py <letra> <x>')

	print '\tTABELA:\n\n\t\ti\txi\tg(xi)\n'
	res = ponto_fixo (sys.argv[1], float (sys.argv[2]))


if __name__ == '__main__':
	main(sys.argv[1:])
```

```
# -*- coding: latin-1 -*-
"""
* Resolução do exercício 6 do capítulo 1.2 (Timothy Sauer. Numerical Analysis. Pearson, 2ª
Edição)
*
* Executado como : python ponto-fixo_1.2-6.py <letra> <x>
*		<letra> -> qual as letras o exercício (a, b ou c)
*		<x> -> representam o chute inicial xo para todas as letras
*				ou
*				  python ponto-fixo_1.2-6.py <letra> <xa> <xb> <xc>
*				  <letra> -> qual das letras do exercícios (a, b ou c)
*		 <xi> -> chute para o exercício i
*
* Parâmetros usados para teste:
*			python ponto-fixo_1.2-6.py a 1
*			python ponto-fixo_1.2-6.py b 0 2 0
*			python ponto-fixo_1.2-2.py c 0 0.5 0
```

```
*
* PS: ga3 e ga2 retornam a mesma raiz!!!
*
"""
import math
import sys


# g(x)'s
def ga1(x):
	return ((6*x + 1)/ 2.0) ** (1/3.0) #essa é a primeira g(x) função (a)
def ga2(x):
	return (2*x**3 - 1)/6.0 #essa é a segunda g(x) função (a)
def ga3(x):
	return (x**3 +1 ) / (3*x**2 - 6)#essa é a terceira g(x) função (a)

def gb1(x):
	return (x - math.exp(x-2)) ** (1/3.0)#essa é a primeira g(x) função (b)
def gb2(x):
	return math.exp(x-2) + x**3 #essa é a segunda g(x) função (b)
def gb3(x):
	return math.exp(x)/(math.exp(2)*(1-x**2)) #essa é a terceira g(x) função (b)

def gc1(x):
	return (6 *x**3 + math.exp(2*x)-1)/5#essa é a primeira g(x) função (c)
def gc2(x):
	return ((1 + 5 *x - math.exp(2*x))/ 6) ** (1/3.0)#essa é a segunda g(x) função (c)
def gc3(x):
	return (math.exp(2*x) - 1)/(5-6*x**2)#essa é a terceira g(x) função (c)


# s(x)'s (derivada de g(x))
def sa1(x):
	return 1/((3*x + 1/2.0) ** (2/3.0))
def sa2(x):
	return x**2
def sa3(x):
	return (x*(x**3 - 6*x - 2))/(3*(x**2 - 2)**2)

def sb1(x):
	return (1 - math.exp(x - 2))/ (3 * (x - math.exp(x-2))**(2/3.0))
def sb2(x):
	return 3 * x**2 + math.exp(x-2)
def sb3(x):
	return (math.exp(x-2) * (-x**2 + 2*x + 1))/(x**2 -1) **2

def sc1(x):
	return (2 * (9*x**2 + math.exp(2*x)))/5
def sc2(x):
	return (5 - 2 * math.exp(2*x))/(3 * 6 ** (1/3.0)* (5*x - math.exp(2*x) + 1) ** (2/3.0))
def sc3(x):
	return (5-18 * x**2)/(-12*x**3 + 10*x + 2)
```

```
MAX = 200 #constante do número máximo de iterações

def ponto_fixo(letra,x,*outros):
	for k in range(3):
		if (letra == 'a'):
			if(k ==0):
				def g(x):
					return ga1(x)
				def s(x):
					return sa1(x)
			elif ( k == 1):
				def g(x):
					return ga2(x)
				def s(x):
					return sa2(x)
				if outros:
					x = outros[0]
			else:
				def g(x):
					return ga3(x)
				def s(x):
					return sa3(x)
				if outros:
					x = outros[1]
		elif (letra == 'b'):
			if(k ==0):
				def g(x):
					return gb1(x)
				def s(x):
					return sb1(x)
			elif ( k == 1):
				def g(x):
					return gb2(x)
				def s(x):
					return sb2(x)
				if outros:
					x = outros[0]
			else:
				def g(x):
					return gb3(x)
				def s(x):
					return sb3(x)
				if outros:
					x = outros[1]
		else:
			if(k ==0):
				def g(x):
					return gc1(x)
				def s(x):
					return sc1(x)
			elif ( k == 1):
```

```
					def g(x):
						return gc2(x)
					def s(x):
						return sc2(x)
					if outros:
						x = outros[0]
				else:
					def g(x):
						return gc3(x)
					def s(x):
						return sc3(x)
					if outros:
						x = outros[1]
			i = 0
			erro = 1
			erro_antigo = 0
			x_novo = x

			while ((erro >.5E-6) and (i < MAX)):
				print '\t\t',i,'\t', x_novo,'\t', g(x_novo)
				x_antigo = x_novo
				x_novo = g(x_novo)
				i = i +1
				erro_antigo = erro
				erro = abs(x_novo - x_antigo)/ (abs(x_novo) + 0.000000001)

			if (i > MAX):
				print 'Limite de 200 iterações atingido'
			#elif (erro == x_novo ):
				print 'g(x) diverge'
			else:
				print '\nA raiz é: ',
				print ("%.8f" % x_novo)
				print 'S por erros: ',
				print("%.4f" % (erro/erro_antigo))
				print 'S por calculo',
				print ("%.4f" % s(x_novo))
				print 'g(x) converge'

def main (argv):

		if (len(sys.argv) < 3):
			sys.exit('Parâmetros errados: ponto_fixo.py <letra> <x1> <x2> <x3>')

		if (len (sys.argv) == 3):
			print '\tTABELA:\n\n\t\ti\txi\tg(xi)\n'
			res = ponto_fixo (sys.argv[1], float (sys.argv[2]))
		else :
			print '\tTABELA:\n\n\t\ti\txi\tg(xi)\n'
			res = ponto_fixo (sys.argv[1], float (sys.argv[2]), float (sys.argv[3]),float
(sys.argv[4]))
```

```
if __name__ == '__main__':
	main(sys.argv[1:])
```

```
# -*- coding: latin-1 -*-
"""
* Resolução do exercício 3 do capítulo 1.4 (Timothy Sauer. Numerical Analysis. Pearson, 2ª
Edição)
*
* Executado como : python newton_1.4-3.py <letra> <x>
*		<letra> -> qual as letras o exercício (a, b ou c)
*		<x> -> representam o chute inicial xo para todas as letras
*
* Parâmetros usados para teste:
*			python newnton_1.4-3.py a
*			python newnton_1.4-3.py b
*			python newnton_1.4-3.py c
*
*
"""
import math
import sys

def fa(x):
	return 27*x**3 + 54*x**2 + 36*x + 8
def dfa(x):
	return 9*(3*x + 2)**2
def ddfa(x):
	return 54*(3*x + 2)

def fb(x):
	return 36*x**4 - 12*x**3 + 37*x**2 - 12*x + 1
def dfb(x):
	return 2*(72*x**3 - 18*x**2 + 37*x - 6)
def dffb(x):
	return 432*x**2 - 72*x + 74

def newton (letra,x):

	if(letra == 'a'):
		def f(x):
			return fa(x)
		def df(x):
			return dfa(x)
	else:
		def f(x):
			return fb(x)
		def df(x):
			return dfb(x)

	aux = 1;
```

```
	i = 0
	print '\t\t',i,'\t', x
	while(aux != 0):
		aux = df(x)
		if(aux != 0):
			x1 = x - f(x)/df(x)
			t = abs(x1 - x)
			i = i + 1
			print '\t\t',i,'\t', x1,'\t', t

			if t == 0:
				break
			x = x1
	return x

def multiplicidade (letra, x):

	if (letra == 'a'):
		def df(x):
			return dfa(x)
		def ddf(x):
			return ddfa(x)
	else:
		def df(x):
			return dfb(x)
		def ddf(x):
			return ddfb(x)

	derivada_primeira = df(x)

	if (derivada_primeira != 0):
		print '\nA raiz tem multiplicidade 1 já que f`(x) = %.12f' % derivada_primeira
	else:
		derivada_segunda = ddf(x)

		print '\nA raiz tem multiplicidade 2 já que a f`(x) = %.12f' % derivada_primeira
		print 'e f``(x) = %f' %derivada_segunda

def modified_newton (letra,x, tolerancia = 0.00000001):
	i = 0
	if(letra == 'a'):
		def f(x):
			return fa(x)
		def df(x):
			return dfa(x)
		def ddf(x):
			return ddfa(x)
	else:
		def f(x):
			return fb(x)
		def df(x):
			return dfb(x)
```

```
		def ddf(x):
			return ddfb(x)

	while(True):
		aux = ((df(x) *df(x)) - (f(x) * ddf(x)))
		if(aux != 0):
			x1 = x - (f(x)* df(x))/((df(x)*df(x)) - (f(x) *ddf(x)))
			t = abs(x1 - x)
			i = i +1
			print '\t\t',i,'\t',x1,'\t',t

			if t < tolerancia:
				break
			x = x1
		else:
			break

	return x


def main (argv):

	if (len(sys.argv) != 3):
		sys.exit('Parâmetros errados: ponto_fixo.py <letra> <x>')

	else :
		print '\tTABELA NEWTON:\n\n\t\ti\txi\terro\n'
		res_n = newton(sys.argv[1], float (sys.argv[2]))

		print'\n\tTABELA NEWTON MODIFICADO:\n\n\t\ti\txi\terro\n'
		res_m = modified_newton(sys.argv[1],float(sys.argv[2]))

		print'\n\nA raiz aproximada ao máximo pelo Método de Newton é ', "%f" % res_n
		multiplicidade(sys.argv[1],res_n)
		print '\nA raiz aproximada pelo Método de Newton Modificado é',"%f\n" %res_m

if __name__ == '__main__':
	main(sys.argv[1:])
```

```
# -*- coding: latin-1 -*-
"""
* Resolução do exercício 5 do capítulo 1.4 (Timothy Sauer. Numerical Analysis. Pearson, 2ª Edição)
*
* Executado como : newton_1.4-5.py <x>
*		<x> -> representam o chute inicial xo
*
* Parâmetros usados para teste:
*			python newton_1.4-5.py 1
*
"""
```

```
import math
import sys
import sympy

def f(x):
	return math.pi*x**2*10 + 2/3.0 * math.pi * x**3 - 400
def df(x):
	return x*(6.28319*x+62.8319)


def newton (x, tolerancia = 0.00001):
	while(True):
		aux = df(x)
		if(aux != 0):
			x1 = x - f(x)/df(x)
			t = abs(x1 - x)
			print ("%.8f" % x1)

			if t < tolerancia:
				break
			x = x1
		else:
			print("Algoritmo interrompido. (divisão por zero) ")
			break
	return x


def main (argv):

	if (len(sys.argv) != 2):
		sys.exit('Parâmetros errados: newton_1.4-5.py <x>')

	res = newton (float(sys.argv[1]))
	print 'Resposta : %0.4f' % res


if __name__ == '__main__':
	main(sys.argv[1:])
```

```
# -*- coding: latin-1 -*-
"""
* Resolução do exercício 5 do capítulo 1.4 (Timothy Sauer. Numerical Analysis. Pearson, 2ª Edição)
*
* Executado como : newton_1.4-6.py <x>
*		<x> -> representam o chute inicial xo
*
* Parâmetros usados para teste:
*			python newton_1.4-6.py 1
*
```

```
"""

import math
import sys
import sympy

def f(x):
	return math.pi*x**2*10 + 2/3.0 * math.pi * x**3 - 60
def df(x):
	return x*(6.28319*x+62.8319)


def newton (x, tolerancia = 0.00001):
	while(True):
		aux = df(x)
		if(aux != 0):
			x1 = x - f(x)/df(x)
			t = abs(x1 - x)
			print ("%.8f" % x1)

			if t < tolerancia:
				break
			x = x1
		else:
			print("Algoritmo interrompido. (divisão por zero) ")
			break
	return x


def main (argv):

	if (len(sys.argv) != 2):
		sys.exit('Parâmetros errados: newton_1.4-6.py <x>')

	res = newton (float(sys.argv[1]))
	print 'Resposta : %0.4f' % res


if __name__ == '__main__':
	main(sys.argv[1:])
```

```
# -*- coding: latin-1 -*-
"""
* Resolução do exercício 5 do capítulo 1.4 (Timothy Sauer. Numerical Analysis. Pearson, 2ª Edição)
*
* Executado como : newton_1.4-7.py <x1> <x2> <x3> <x4>
*		<x> -> representam o chute inicial xo
*
* Parâmetros usados para teste:
*			python newton_1.4-7.py -1 1.5 0
```

```
*
"""

import math
import sys
import sympy
import numpy as np
import matplotlib.pyplot as plt

def graph ():
	x = np.arange(-2.0,2.1,0.5)
	y = f(x)
	plt.plot(x,y)
	plt.show()

def f(x):
	return np.exp(np.sin(x)**3) + x**6 - 2*x**4 - x**3 - 1
def df(x):
	return (6*x**3- 8*x - 3)*x**2 + 3*np.exp(np.sin(x)**3)*(np.sin(x)**2)*np.cos(x)
def ddf(x):
	return (6*x*(5*x**3 - 4*x-1) - 3*np.exp(np.sin(x))**3)*(np.sin(x)**3) +
3*np.exp(np.sin(x)**3)*(3*(np.sin(x)**3) + 2)*np.sin(x)*np.cos(x)**2
def dddf(x):
	return 3*(2*(20*x**3 - 8*x - 1)*np.sin(x)**3 - np.sin(x)**2*(6*x*(-5*x**3 + 4*x +1) + 9
* np.exp(np.sin(x)**3) * np.sin*(x)**3 + 7*np.exp(np.sin(x)**3))*np.cos(x)
	      + np.exp(np.sin(x)**3)(9*np.sin(x)**6 + 18*np.sin(x)**3+2)*np.cos(x)**3)

def newton (x, tolerancia = 0.000001):
	while(True):
		aux = df(x)
		if(aux != 0):
			x1 = x - f(x)/df(x)
			t = abs(x1 - x)/(abs(x1) + 0.000001)


			if t < tolerancia:
				break
			x = x1
		else:
			print("Algoritmo interrompido. (divisão por zero) ")
			break
	return x

def multiplecidade():
	print ok


def multiplicidade_quadrada (x):

	derivada_primeira = df(x)
	derivada_segunda = ddf(x)
```

```
	if (derivada_primeira != 0):
		return 1

def multiplicidade_linear(x):
	derivada_segunda = ddf(x)
	derivada_terceira = dddf(x)

	if (derivada_segunda !=0):
		return 2
	else:
		return 3

def main (argv):

	if (len(sys.argv) != 4):

		sys.exit('Parâmetros errados: newton_1.4-7.py <x1> <x2> <x3>')

	graph()

	raiz1 = newton(float(sys.argv[1]))
	raiz2 = newton(float(sys.argv[2]))
	raiz3 = newton(float(sys.argv[3]))

	print 'Raizes: '

	for i in range(1,4):

		if i == 1:
			aux = raiz1
		elif i ==2:
			aux = raiz2
		else:
			aux = raiz3


		mul = multiplicidade_quadrada(aux)
		if mul == 1 :
			print "%.6f" %aux,
			print', convergência quadrada;'
		else:
			print aux,
			print',convergência linear e m=',mul

if __name__ == '__main__':
	main(sys.argv[1:])
```

```
# -*- coding: latin-1 -*-
"""
* Resolução do exercício 5 do capítulo 1.4 (Timothy Sauer. Numerical Analysis. Pearson, 2ª
Edição)
```

```
*
* Executado como : newton_1.4-10.py <x1> <x2> <x3> <x4> <x5>
*             <xi> -> representam o chute inicial xi
*
* Parâmetros usados para teste:
*                     python newton_1.4-10.py 0 1 -1.5 0.8 -0.5
*
"""

import math
import sys
import sympy
import numpy as np
import matplotlib.pyplot as plt

def graph ():
    x = np.arange(-2.0,2.1,0.5)
    y = f(x)
    plt.plot(x,y)
    plt.show()

def f(x):
    return 54*x**6 + 45*x**5 - 102*x**4- 69*x**3 + 35*x**2 + 16*x - 4
def df(x):
    return 324*x**5 + 225*x**4 - 408*x**3 - 207*x**2 + 70*x + 16
def ddf(x):
    return 2*(810*x**4 + 450*x**3 - 612*x**3 + 35*x**2 + 16*x)


def newton (x, tolerancia = 0.000000001):
    while(True):
        aux = df(x)
        if(aux != 0):
            x1 = x - f(x)/df(x)
            t = abs(x1 - x)/(abs(x1) + 0.000001)


            if t < tolerancia:
                break
            x = x1
        else:
            print("Algoritmo interrompido. (divisão por zero) ")
            break
    return x

def multiplicidade(x):

    derivada_primeira = df(x)

    if (derivada_primeira != 0):
        return 1
    elif (derivada_segunda !=0):
```

```
        return 2
    elif (derivada_terceira !=0):
        return 3
    else:
        return 0

def main (argv):

    if (len(sys.argv) != 6):

        sys.exit('Parâmetros errados: newton_1.4-7.py <x1> <x2> <x3> <x4> <x5>')

    graph()

    raiz1 = newton(float(sys.argv[1]))
    raiz2 = newton(float(sys.argv[2]))
    raiz3 = newton(float(sys.argv[3]))
    raiz4 = newton(float(sys.argv[4]))
    raiz5 = newton(float(sys.argv[5]))

    print '\nRaizes: '

    for i in range(1,6):

        if i == 1:
            aux = raiz1
        elif i ==2:
            aux = raiz2
        elif i == 3:
            aux = raiz3
        elif i == 4:
            aux = raiz4
        else:
            aux = raiz5


        mul = multiplicidade(aux)
        if mul == 1 :
            print "%.6f" %aux,
            print', convergência quadrada;'
        else:
            print aux,
            print',convergência linear e m=',mul

if __name__ == '__main__':
    main(sys.argv[1:])
```

```
# -*- coding: latin-1 -*-
"""
* Resolução do exercício 5 do capítulo 1.4 (Timothy Sauer. Numerical Analysis. Pearson, 2ª
Edição)
```

```
*
* Executado como : newton_1.4-11.py
*
* Parâmetros usados para teste:
*                       python newton_1.4-11.py
*
"""

import math
import sys
import sympy


def f(x):
	return (15 + (1.36 / x**2)) * (x - 0.003183) - (0.0820578*320)
def df(x):
	return (15*(x**3 - 0.0906667*x + 0.000577184))/x**3

def newton (x, tolerancia = 0.0001):
	while(True):
		aux = df(x)
		if(aux != 0):
			x1 = x - f(x)/df(x)
			t = abs(x1 - x)/(abs(x1) + 0.000001)


			if t < tolerancia:
				break
			x = x1
		else:
			print("Algoritmo interrompido. (divisão por zero) ")
			break
	return x


def main (argv):

	x = (0.0820578*320)/15
	raiz = newton(x)
	print "\nChute inicial = %f" % x,
	print " e raiz = %.6f"%raiz



if __name__ == '__main__':
	main(sys.argv[1:])
```

Lista 3

```
# -*- coding: latin-1 -*-
"""
* Resolução do exercício 5 do capítulo 1.4 (Timothy Sauer. Numerical Analysis. Pearson, 2ª
Edição)
*
* Executado como : newton_1.4-11.py
*
* Parâmetros usados para teste:
*                       python newton_1.4-11.py
*
"""

import math
import sys
import sympy


def f(x):
        return (15 + (1.36 / x**2)) * (x - 0.003183) - (0.0820578*320)
def df(x):
        return (15*(x**3 - 0.0906667*x + 0.000577184))/x**3

def newton (x, tolerancia = 0.0001):
        while(True):
                aux = df(x)
                if(aux != 0):
                        x1 = x - f(x)/df(x)
                        t = abs(x1 - x)/(abs(x1) + 0.000001)


                        if t < tolerancia:
                                break
                        x = x1
                else:
                        print("Algoritmo interrompido. (divisão por zero) ")
                        break
        return x


def main (argv):

        x = (0.0820578*320)/15
        raiz = newton(x)
        print "\nChute inicial = %f" % x,
        print " e raiz = %.6f"%raiz



if __name__ == '__main__':
        main(sys.argv[1:])
```

```
# -*- coding: latin-1 -*-
"""
* Resolução do exercício 5 do capítulo 1.4 (Timothy Sauer. Numerical Analysis. Pearson, 2ª
Edição)
*
* Executado como : newton_1.4-11.py
*
* Parâmetros usados para teste:
*                       python newton_1.4-11.py
*
"""

import math
import sys
import sympy


def f(x):
        return (15 + (1.36 / x**2)) * (x - 0.003183) - (0.0820578*320)
def df(x):
        return (15*(x**3 - 0.0906667*x + 0.000577184))/x**3

def newton (x, tolerancia = 0.0001):
        while(True):
                aux = df(x)
                if(aux != 0):
                        x1 = x - f(x)/df(x)
                        t = abs(x1 - x)/(abs(x1) + 0.000001)


                        if t < tolerancia:
                                break
                        x = x1
                else:
                        print("Algoritmo interrompido. (divisão por zero) ")
                        break
        return x


def main (argv):

        x = (0.0820578*320)/15
        raiz = newton(x)
        print "\nChute inicial = %f" % x,
        print " e raiz = %.6f"%raiz



if __name__ == '__main__':
```

```
        main(sys.argv[1:])
```

```
# -*- coding: latin-1 -*-
"""
* Resolução do exercício 5 do capítulo 1.4 (Timothy Sauer. Numerical Analysis. Pearson, 2ª
Edição)
*
* Executado como : secante_1.5-7.py <x1> <x2> <x3> <x4>
*               <x> -> representam o chute inicial xo
*
* Parâmetros usados para teste:
*                       python secante_1.5-7.py -1 1.5 0
*
"""

import math
import sys
import sympy
import numpy as np
import matplotlib.pyplot as plt

def graph ():
        x = np.arange(-2.0,2.1,0.5)
        y = f(x)
        plt.plot(x,y)
        plt.show()

def f(x):
        return 54*x**6 + 45*x**5 - 102*x**4 -69*x**3 + 35*x**2 + 16*x - 4

def secante(x0,x1, tolerancia=0.00000001, NMAX=200):

        n=1
        while n<=NMAX:
                x2 = x1 - (f(x1)*(x1-x0))/(f(x1)-f(x0))

                t = abs(x2 - x1)
                if t < tolerancia:
                        return x2
                else:
                        x0 = x1
                        x1 = x2
        return False

def main (argv):

        graph()

        print "Raizes:"
        raiz1 = secante (1.0,-1.0)
        print "(1) %.6f com os chutes iniciais 1 e -1" %raiz1
```

```
        raiz2 = secante (0.0,-1.0)
        print "(2) %.6f com os chutes iniciais 0 e -1" %raiz2
        raiz3 = secante (1.0,2.0)
        print "(3) %.6f com os chutes iniciais 1 e 2" %raiz3
        raiz4 = secante (0.0,0.4)
        print "(4) %.6f com os chutes iniciais 0 e 0.4" %raiz4
        raiz5 = secante (-2.0,-1.5)
        print "(5) %.6f com os chutes iniciais -2 e -1.5" %raiz5
if __name__ == '__main__':
    main(sys.argv[1:])
```

Lista 4

```
# -*- coding: latin-1 -*-
"""
* Resolução da Decomposição LU
*
* Executado como : decomposicao_LU.py
*
* Parâmetros usados para teste:
*                       python decomposicao_LU.py
*
"""
import math
import sys
import pprint


def mult(M1, M2):

    truple = zip(*M2)

    return [[sum(em1*em2 for em1,em2 in zip(m1,m2)) for m2 in truple] for m1 in M1]

def permut(m):

    n = len(m)
    identidade = [[float(i == j) for i in xrange(n)] for j in xrange(n)]

    for j in xrange(n):
        row = max(xrange(j, n), key=lambda i: abs(m[i][j]))
        if j != row:
            identidade[j], identidade[row] = identidade[row], identidade[j]

    return identidade

def lu(A):

    n = len(A)

    L = [[0.0] * n for i in xrange(n)]
    U = [[0.0] * n for i in xrange(n)]
```

```
    p = permut(A)
    pA = mult(p, A)

    for j in xrange(n):

        L[j][j] = 1.0

        for i in xrange(j+1):
            s1 = sum(U[k][j] * L[i][k] for k in xrange(i))
            U[i][j] = pA[i][j] - s1

        for i in xrange(j, n):
            s2 = sum(U[k][j] * L[i][k] for k in xrange(j))
            L[i][j] = (pA[i][j] - s2) / U[j][j]

    return (L, U, p)

def main (argv):
    A = [[7,3,-1,2],[3,8,1,-4],[-1,1,4,-1],[2,-4,-1,6]]

    L,U,p = lu(A)

    print "A: "
    pprint.pprint(A)

    print "P: "
    pprint.pprint(p)

    print "L: "
    pprint.pprint(L)

    print "U: "
    pprint.pprint(U)


    """for part in lu(A):
        pprint(part, widentidadeth=19)
    print
    print
    B = [[11,9,24,2],[1,5,2,6],[3,17,18,1],[2,5,7,1]]
    for part in lu(B):
        pprint(part)
    print"""

if __name__ == '__main__':
    main(sys.argv[1:])
```

```
# -*- coding: latin-1 -*-
"""
```

```
* Resolução do método da Decomposição de Cholesky
*
* Executado como : decomposicao_cholesky.py
*
* Parâmetros usados para teste:
*                     python decomposicao_cholesky.py
*
"""
import math
import sys
import pprint


def cholesky(A):

    L = [[0.0] * len(A) for _ in xrange(len(A))]

    for i in xrange(len(A)):
        for j in xrange(i+1):
            s = sum(L[i][k] * L[j][k] for k in xrange(j))
            L[i][j] = math.sqrt(A[i][i] - s) if (i == j) else \
                      (1.0 / L[j][j] * (A[i][j] - s))
    return L

def main (argv):
    m1 = [[4, 2, -4],
          [2, 10,  4],
          [-4,  4, 9]]
    g = cholesky(m1)
    gt = zip(*g)

    print "M1:"
    pprint.pprint(m1)
    print "G:"
    pprint.pprint(g)
    print "Gt:"/
    pprint.pprint(gt)

if __name__ == '__main__':
    main(sys.argv[1:])
```

```
# -*- coding: latin-1 -*-
"""
* Resolução do método da Resolução de Sistemas Triangulares Superiores
*
* Executado como : sistema_triangular_superior.py
*
* Parâmetros usados para teste:
*                     python sistema_triangular_superior.py
*
"""
```

```
import math
import sys
import pprint
import numpy as np

def triangular_superior (U,b):
    n = np.size(b)
    x = []
    a = 0.
    while a<n:
        x.append(0.)
        a = a +1

    x[n-1] = b[n-1]/U[n-1][n-1]
    i = n - 1
    while i >= 0:
        s = b[i]
        j = i +1
        while j < n:
            s = s - U[i][j]*x[j]
            j = j + 1
        x[i] = s/U[i][i]
        i = i -1

    return x

def main (argv):
    U = [[5.,2.,1.],
        [0.,-1/5.,17/5.],
        [0.,0.,13.]]

    b = [0.,-7.,-26.]

    res = triangular_superior(U,b)
    res = ["%.2f" % r for r in res]
    print "X:"
    pprint.pprint(res)

if __name__ == '__main__':
    main(sys.argv[1:])
```

Lista 5

```
# -*- coding: latin-1 -*-
"""
* Resolução do exercício 5.1 (Neide Maria B. Franco. Cálculo Numérico. Pearson. 1ª Edição.)
*
* Executado como : franco_5-1.py
*
* Parâmetros usados para teste:
*                         python franco_5-1.py
*
"""
```

```
import sys
from pprint import pprint
from numpy import array, zeros, diag, diagflat, dot

def jacobi(A,b,N=25,x=None):

    #Cria um chute inicial se necessário
    if x is None:
        x = zeros(len(A[0]))


    D = diag(A)
    R = A - diagflat(D)

    # Itera por N vezes
    for i in range(N):
        x = (b - dot(R,x)) / D
    return x

def main (argv):

    A = array([[4,-1,0,-1,0,0],
              [-1,4,-1,0,-1,0],
              [0,-1, 4,0, 0,-1],
              [-1,0,0,4,-1,0],
              [0,-1,0,-1,4,-1],
              [0,0,-1,0,-1,4]])

    b = array([100,0,0,100,0,0])

    guess = None

    sol = jacobi(A,b,N=25,x=guess)

    print "A:"
    pprint(A)

    print "b:"
    pprint(b)

    print "x:"
    pprint(sol)

if __name__ == '__main__':
    main(sys.argv[1:])
```

```
# -*- coding: latin-1 -*-
"""
* Resolução do exercício 5.2 (Neide Maria B. Franco. Cálculo Numérico. Pearson. 1ª Edição.)
*
```

```
* Executado como : franco_5-2.py.py
*
* Parâmetros usados para teste:
*                      python franco_5-2.py.py
*
"""

import sys
import numpy as np
from scipy.linalg import solve


def gauss(A, b, x, n):

    L = np.tril(A)
    U = A - L
    m = len(x)
    x_novo = x[1]
    for i in range(n):
        # x^(k+1) = L*^(-1)*(b-Ux^(k))
        x = np.dot(np.linalg.inv(L), b - np.dot(U, x))

        x_antigo = x_novo
        for k in range(m):
            if x[k] > x_novo:
                x_novo = x[k]
        t = abs(x_novo - x_antigo)/abs(x_novo)
        if(t < 0.001):
            break
    return x

def main (argv):
    A = np.array([[20,-10,-4],[-10,25,-5],[-4,5,10]])
    b = [26,0,7]
    x = [1, 1, 1]
    n = 20
    print gauss(A, b, x, n)



if __name__ == '__main__':
    main(sys.argv[1:])
```

```
# -*- coding: latin-1 -*-
"""
* Resolução do exercício 5.2 (Neide Maria B. Franco. Cálculo Numérico. Pearson. 1ª Edição.)
*
* Executado como : franco_5-2.py.py
*
* Parâmetros usados para teste:
*                      python franco_5-2.py.py
```

```
*
"""

import sys
import numpy as np
from scipy.linalg import solve


def gauss(A, b, x, n):

    L = np.tril(A)
    U = A - L
    m = len(x)
    x_novo = x[1]
    for i in range(n):
        # x^(k+1) = L*^(-1)*(b-Ux^(k))
        x = np.dot(np.linalg.inv(L), b - np.dot(U, x))

        x_antigo = x_novo
        for k in range(m):
            if x[k] > x_novo:
                x_novo = x[k]
        t = abs(x_novo - x_antigo)/abs(x_novo)
        if(t < 0.001):
            break
    return x

def main (argv):
    A = np.array([[20,-10,-4],[-10,25,-5],[-4,5,10]])
    b = [26,0,7]
    x = [1, 1, 1]
    n = 20
    print gauss(A, b, x, n)



if __name__ == '__main__':
    main(sys.argv[1:])
```

```
# -*- coding: latin-1 -*-
"""
* Resolução do exercício 5.5 (Neide Maria B. Franco. Cálculo Numérico. Pearson. 1ª Edição.)
*
* Executado como : franco_5.8.py
*
* Parâmetros usados para teste:
*                         python franco_5.8.py
*
"""

import sys
```

```
from pprint import pprint
from numpy import array, zeros, diag, diagflat, dot
from scipy.linalg import solve

def jacobi(A,b,N=25,x=None):

    #Cria um chute inicial se necessário
    if x is None:
        x = zeros(len(A[0]))


    D = diag(A)
    R = A - diagflat(D)

    # Itera por N vezes
    for i in range(N):
        x = (b - dot(R,x)) / D
    return x

def main (argv):

    A = array([[10,   0,  0, 100,  0, 0],
               [10,-100,  0,   0,100, 0],
               [ 0,   0,100,-100,  0, 0],
               [ 1,   1,  0,   0,  0,-1],
               [-1,   0,  0,   1,  1, 0],
               [ 0,   1, -1,   0,  1, 0]])

    b = array([20,0,0,0,0,0])

    guess = None

    sol = jacobi(A,b,N=25,x=guess)

    print "A:"
    pprint(A)

    print "b:"
    pprint(b)

    print "x:"
    pprint(sol)


if __name__ == '__main__':
    main(sys.argv[1:])
```

Lista 7

```
# -*- coding: latin-1 -*-
"""
```

```
* Resolução do exercício 5.9 (Neide Maria B. Franco. Cálculo Numérico. Pearson. 1ª Edição.)
*
* Executado como : franco_5-9.py
*
* Parâmetros usados para teste:
*                       python franco_5-9.py
*
"""

import sys
from pprint import pprint
from numpy import array, zeros, diag, diagflat, dot,linalg

def converge_dominante(A):

    n = len(A)
    soma = 0
    for i in range(n):
        j = i
        if j != 0:
            k = j-1
            while k >= 0:
                soma = abs(A[i][k]) + soma
                k = k -1

        if j != n-1:
            k = j+1
            while k <n:
                soma = abs(A[i][k]) + soma
                k = k+1
        if soma >=  A[i][j]:
            return False
            break
        soma = 0
    return True

def main (argv):

    A = array([[1,-0.5,0],[-0.5,1,-0.5],[0,-0.5,1]])
    B = array([[1,0.5,0],[0.5,1,0.5],[0,0.5,1]])
    bol = converge_dominante(A)
    bol2 = converge_dominante(B)

    print "A:"
    pprint(A)

    if bol:
        print "\n A matriz A é convergente."
    else:
        print "\n A matriz A não é convergente."

    print "B:"
```

```
    pprint(B)

    if bol2:
        print "\n A matriz B é convergente."
    else:
        print "\n A matriz B não é convergente."



if __name__ == '__main__':
    main(sys.argv[1:])
```

```
# -*- coding: latin-1 -*-
"""
* Resolução do exercício 5.10 (Neide Maria B. Franco. Cálculo Numérico. Pearson. 1ª Edição.)
*
* Executado como : franco_5-10.py
*
* Parâmetros usados para teste:
*                       python franco_5-10.py
*
"""

import sys
from pprint import pprint
from numpy import array, zeros, diag, diagflat, dot,linalg,tril

def sassenfeld(A):

    n = len(A)
    soma = 0
    B = [1] * len(A)

    for i in range(n):
        j = i
        if j != 0:
            k = j-1
            while k >= 0:
                soma = abs(A[i][k])*B[k] + soma
                k = k -1

        if j != n-1:
            k = j+1
            while k <n:
                soma = abs(A[i][k])*B[k] + soma
                k = k+1
        B[i] = soma/float(abs(A[i][j]))
        if B[i] > 1:
            return False
            break
        soma = 0
```

```
    return True

def gauss(A, b, x, n):

    L = tril(A)
    U = A - L
    for i in range(n):
        # x^(k+1) = L*^(-1)*(b-Ux^(k))
        x = dot(linalg.inv(L), b - dot(U, x))

    return x

def main (argv):

    A = array([[10,-1,4],[1,10,9],[2,-3,-10]])
    b = [5,2,9]
    x = [1,1,1]
    n = 5

    print "A:"
    pprint(A)

    bol = sassenfeld(A)
    if bol:
        print "\n A matriz A é convergente por Sassenfeld."
    else:
        print "\n A matriz A não é convergente por Sassenfeld."

    res = gauss(A,b,x,n)

    print "x: "
    pprint(res)


if __name__ == '__main__':
    main(sys.argv[1:])
```

```
# -*- coding: latin-1 -*-
"""
* Resolução do exercício 5.11 (Neide Maria B. Franco. Cálculo Numérico. Pearson. 1ª Edição.)
*
* Executado como : franco_5-11.py
*
* Parâmetros usados para teste:
*                    python franco_5-11.py
*
"""

import sys
from pprint import pprint
from numpy import array, zeros, diag, diagflat, dot,linalg,tril
```

```
def sassenfeld(A):

    n = len(A)
    soma = 0
    B = [1] * len(A)

    for i in range(n):
        j = i
        if j != 0:
            k = j-1
            while k >= 0:
                soma = abs(A[i][k])*B[k] + soma
                k = k -1

        if j != n-1:
            k = j+1
            while k <n:
                soma = abs(A[i][k])*B[k] + soma
                k = k+1
        B[i] = soma/float(abs(A[i][j]))
        if B[i] > 1:
            return False
            break
        soma = 0
    return True

def gauss(A, b, x, n):

    L = tril(A)
    U = A - L
    for i in range(n):
        # x^(k+1) = L*^(-1)*(b-Ux^(k))
        x = dot(linalg.inv(L), b - dot(U, x))

    return x

def main (argv):

    A = array([[50,-1,4],[1,50,9],[2,-3,-50]])
    b = [45,42,49]
    x = [1,1,1]
    n = 3

    print "A:"
    pprint(A)

    bol = sassenfeld(A)
    if bol:
        print "\n A matriz A é convergente por Sassenfeld."
    else:
        print "\n A matriz A não é convergente por Sassenfeld."
```

```
    res = gauss(A,b,x,n)

    print "x: "
    pprint(res)


if __name__ == '__main__':
    main(sys.argv[1:])
```

```
# -*- coding: latin-1 -*-
"""
* Resolução do exercício 5.12 (Neide Maria B. Franco. Cálculo Numérico. Pearson. 1ª Edição.)
*
* Executado como : franco_5-12.py
*
* Parâmetros usados para teste:
*                       python franco_5-12.py
*
"""

import sys
from pprint import pprint
from numpy import array, zeros, diag, diagflat, dot,linalg,tril
from scipy.linalg import solve

def sassenfeld(A):

    n = len(A)
    soma = 0
    B = [1] * len(A)

    for i in range(n):
        j = i
        if j != 0:
            k = j-1
            while k >= 0:
                soma = abs(A[i][k])*B[k] + soma
                k = k -1

        if j != n-1:
            k = j+1
            while k <n:
                soma = abs(A[i][k])*B[k] + soma
                k = k+1
        B[i] = soma/float(abs(A[i][j]))

        if B[i] > 1:
            return False
            break
        soma = 0
```

```
    return True

def gauss(A, b, x, n):

    L = tril(A)
    U = A - L
    for i in range(n):
        # x^(k+1) = L*^(-1)*(b-Ux^(k))
        x = dot(linalg.inv(L), b - dot(U, x))

    return x

def main (argv):

    AI = array([[5,2,1],[2,4,1],[2,2,4]])
    bI = [0,2,1]
    x = [1,1,1]
    n = 5

    print "(I):"
    pprint(AI)

    bol = sassenfeld(AI)
    if bol:
        print "\n A matriz (I) é convergente por Sassenfeld."
        res = gauss(AI,bI,x,n)

        print "x para (I): "
        pprint(res)

    else:
        print "\n A matriz (I) não é convergente por Sassenfeld."


    AII = array([[5,4,1],[3,4,1],[3,3,6]])
    bII = [2,2,-9]
    x = [1,1,1]
    n = 5

    print "(II):"
    pprint(AII)

    bol = sassenfeld(AII)
    if bol:
        print "\n A matriz (II) é convergente por Sassenfeld."
        res = gauss(AII,bII,x,n)

        print "x para (II): "
        pprint(res)

    else:
```

```
    print "\n A matriz (II) não é convergente por Sassenfeld."

if __name__ == '__main__':
  main(sys.argv[1:])
```

```
# -*- coding: latin-1 -*-
"""
* Resolução do exercício 5.13 (Neide Maria B. Franco. Cálculo Numérico. Pearson. 1ª Edição.)
*
* Executado como : franco_5-13.py
*
* Parâmetros usados para teste:
*                       python franco_5-13.py
*
"""

import sys
from pprint import pprint
from numpy import array, zeros, diag, diagflat, dot,linalg,tril
from scipy.linalg import solve

def sassenfeld(A):

  n = len(A)
  soma = 0
  B = [1] * len(A)

  for i in range(n):
    j = i
    if j != 0:
      k = j-1
      while k >= 0:
        soma = abs(A[i][k])*B[k] + soma
        k = k -1

    if j != n-1:
      k = j+1
      while k <n:
        soma = abs(A[i][k])*B[k] + soma
        k = k+1
    B[i] = soma/float(abs(A[i][j]))
    if B[i] > 1:
      return False
      break
    soma = 0
  return True

def gauss(A, b, x, n):

  L = tril(A)
  U = A - L
```

```
    for i in range(n):
        # x^(k+1) = L*^(-1)*(b-Ux^(k))
        x_novo = dot(linalg.inv(L), b - dot(U, x))
        erro = (linalg.norm(x - x_novo))/linalg.norm(x_novo)
        if erro < 0.01:
            return x_novo
        x = x_novo

    return x

def main (argv):

    A = array([[2,-1,0,0],[-1,2,-1,0],[0,-1,2,-1],[0,0,-1,1]])
    b = [1,1,1,1]
    x = [1,1,1,1]
    n = 200

    print "A:"
    pprint(A)

    bol = sassenfeld(A)
    if bol:
        print "\n A matriz A é convergente por Sassenfeld."
    else:
        print "\n A matriz A não é convergente por Sassenfeld."

    res = gauss(A,b,x,n)

    print "x: "
    pprint(res)

    #print solve(A,b)

if __name__ == '__main__':
    main(sys.argv[1:])
```

```
# -*- coding: latin-1 -*-
"""
* Resolução do exercício 5.18 (Neide Maria B. Franco. Cálculo Numérico. Pearson. 1ª Edição.)
*
* Executado como : franco_5-18.py
*
* Parâmetros usados para teste:
*                    python franco_5-18.py
*
"""

import sys
from pprint import pprint
from numpy import array, zeros, diag, diagflat, dot,linalg,tril
from scipy.linalg import solve
```

```
def sassenfeld(A):

    n = len(A)
    soma = 0
    B = [1] * len(A)

    for i in range(n):
        j = i
        if j != 0:
            k = j-1
            while k >= 0:
                soma = abs(A[i][k])*B[k] + soma
                k = k -1

        if j != n-1:
            k = j+1
            while k <n:
                soma = abs(A[i][k])*B[k] + soma
                k = k+1
        B[i] = soma/float(abs(A[i][j]))
        if B[i] > 1:
            return False
            break
        soma = 0
    return True


def gauss(A, b, x, n):

    L = tril(A)
    U = A - L
    for i in range(n):
        # x^(k+1) = L*^(-1)*(b-Ux^(k))
        x_novo = dot(linalg.inv(L), b - dot(U, x))
        erro = (linalg.norm(x - x_novo))/linalg.norm(x_novo)
        if erro < 0.01:
            return x_novo
        x = x_novo

    return x

def main (argv):

    A = array([[2,-1,0,0],[-1,2,-1,0],[0,-1,2,-1],[0,0,-1,2]])
    b = [2,1,9,11]
    x = [1,1,1,1]
    n = 200

    print "A:"
    pprint(A)
```

```
    bol = sassenfeld(A)
    if bol:
        print "\n A matriz A é convergente por Sassenfeld."
    else:
        print "\n A matriz A não é convergente por Sassenfeld."

    res = gauss(A,b,x,n)

    print "x: "
    pprint(res)

    #print solve(A,b)

if __name__ == '__main__':
    main(sys.argv[1:])
```

```
# -*- coding: latin-1 -*-
"""
* Resolução do exercício 5.18 (Neide Maria B. Franco. Cálculo Numérico. Pearson. 1ª Edição.)
*
* Executado como : franco_5-18.py
*
* Parâmetros usados para teste:
*                       python franco_5-18.py
*
"""

import sys
from pprint import pprint
from numpy import array, zeros, diag, diagflat, dot,linalg,tril
from scipy.linalg import solve

def sassenfeld(A):

    n = len(A)
    soma = 0
    B = [1] * len(A)

    for i in range(n):
        j = i
        if j != 0:
            k = j-1
            while k >= 0:
                soma = abs(A[i][k])*B[k] + soma
                k = k -1

        if j != n-1:
            k = j+1
            while k <n:
                soma = abs(A[i][k])*B[k] + soma
                k = k+1
```

```
        B[i] = soma/float(abs(A[i][j]))
        if B[i] > 1:
            return False
            break
        soma = 0
    return True


def gauss(A, b, x, n):

    L = tril(A)
    U = A - L
    for i in range(n):
        # x^(k+1) = L*^(-1)*(b-Ux^(k))
        x_novo = dot(linalg.inv(L), b - dot(U, x))
        erro = (linalg.norm(x - x_novo))/linalg.norm(x_novo)
        if erro < 0.01:
            return x_novo
        x = x_novo

    return x

def main (argv):

    A = array([[2,-1,0,0],[-1,2,-1,0],[0,-1,2,-1],[0,0,-1,2]])
    b = [2,1,9,11]
    x = [1,1,1,1]
    n = 200

    print "A:"
    pprint(A)

    bol = sassenfeld(A)
    if bol:
        print "\n A matriz A é convergente por Sassenfeld."
    else:
        print "\n A matriz A não é convergente por Sassenfeld."

    res = gauss(A,b,x,n)

    print "x: "
    pprint(res)

    #print solve(A,b)

if __name__ == '__main__':
    main(sys.argv[1:])
```

```
# -*- coding: latin-1 -*-
"""
* Resolução do exercício 5.20 (Neide Maria B. Franco. Cálculo Numérico. Pearson. 1ª Edição.)
```

```
*
* Executado como : franco_5-20.py
*
* Parâmetros usados para teste:
*                       python franco_5-20.py
*
"""

import sys
from pprint import pprint
from numpy import array, zeros, diag, diagflat, dot,linalg,divide,amax

def converge_dominante(A):

    n = len(A)
    soma = 0
    for i in range(n):
        j = i
        if j != 0:
            k = j-1
            while k >= 0:
                soma = abs(A[i][k]) + soma
                k = k -1

        if j != n-1:
            k = j+1
            while k <n:
                soma = abs(A[i][k]) + soma
                k = k+1
        if soma >=  A[i][j]:
            return False
            break
        soma = 0
    return True


def converge_linha(A):

    n = len(A)
    diag = A.diagonal()
    x = [[0.0] * len(A) for _ in xrange(len(A))]

    for i in range(n):
        for j in range(n):
            x[i][j] = A[i][j]/float(diag[i])

    B = [0.0] * n
    soma = 0

    for i in range(n):
        j = i
        if j != 0:
```

```
            k = j-1
            while k >= 0:
                soma = abs(x[i][k])+ soma
                k = k -1

        if j != n-1:
            k = j+1
            while k <n:
                soma = abs(x[i][k]) + soma
                k = k+1
        B[i] = soma
        soma = 0
    maior = amax(B)

    if maior < 1:
        return True
    else:
        return False

def converge_coluna(A):

    n = len(A)
    diag = A.diagonal()
    x = [[0.0] * len(A) for _ in xrange(len(A))]

    for i in range(n):
        for j in range(n):
            x[i][j] = A[i][j]/float(diag[i])

    B = [0.0] * n
    soma = 0

    for i in range(n):
        j = i
        if j != 0:
            k = j-1
            while k >= 0:
                soma = abs(x[k][i])+ soma
                k = k -1

        if j != n-1:
            k = j+1
            while k <n:
                soma = abs(x[k][i]) + soma
                k = k+1
        B[i] = soma
        soma = 0
    maior = amax(B)
    print B
    if maior < 1:
        return True
    else:
```

```
        return False


def main (argv):

    A = array([[1,2,-2],[1,1,1],[2,2,1]])

    bol= converge_linha(A)

    print "A:"
    pprint(A)
    if bol:
        print "\n A matriz A é convergente pelo critério das linhas."
    else:
        print "\n A matriz A não é convergente pelo critério das linhas."
        bol = converge_coluna(A)
        if bol:
            print "\n A matriz A é convergente pelo critério das colunas."
        else:
            print "\n A matriz A não é convergente pelo critério das colunas."
            bol = converge_dominante(A)
            if bol:
                print "\n A matriz A é convergente porque é matriz dominante."
            else:
                print "\n A matriz A não é matriz dominante."


if __name__ == '__main__':
    main(sys.argv[1:]
```

```
# -*- coding: latin-1 -*-
"""
* Resolução do exercício 5.21 (Neide Maria B. Franco. Cálculo Numérico. Pearson. 1ª Edição.)
*
* Executado como : franco_5-21.py
*
* Parâmetros usados para teste:
*                       python franco_5-21.py
*
"""

import sys
from pprint import pprint
from numpy import array, zeros, diag, diagflat, dot,linalg,divide,amax

def converge_linha(A):

    n = len(A)
    diag = A.diagonal()
    x = [[0.0] * len(A) for _ in xrange(len(A))]

    for i in range(n):
        for j in range(n):
```

```
            x[i][j] = A[i][j]/float(diag[i])

    B = [0.0] * n
    soma = 0

    for i in range(n):
        j = i
        if j != 0:
            k = j-1
            while k >= 0:
                soma = abs(x[i][k])+ soma
                k = k -1

        if j != n-1:
            k = j+1
            while k <n:
                soma = abs(x[i][k]) + soma
                k = k+1
        B[i] = soma
        soma = 0
    maior = amax(B)

    if maior < 1:
        return True
    else:
        return False

def main (argv):

    A = array([[20,3,1],[18,20,1],[1,4,6]])

    bol= converge_linha(A)

    print "A:"
    pprint(A)
    if bol:
        print "\n A matriz A é convergente pelo critério das linhas."
    else:
        print "\n A matriz A não é convergente pelo critério das linhas."


if __name__ == '__main__':
    main(sys.argv[1:])
```

Lista 8

```
# -*- coding: latin-1 -*-
"""
* Resolução do exercício 10.1 (Neide Maria B. Franco. Cálculo Numérico. Pearson. 1ª Edição.)
*
* Executado como : franco_10.1.py
```

```
*
* Parâmetros usados para teste:
*                          python franco_10.1.py
*
"""

import sys
import math
from sympy import *
from sympy.matrices import *
import numpy as np

def lagrange(pontos, valor=None):
   x = Symbol("x")
   L = zeros(1, pontos.shape[0])
   i = 0

   for p in pontos:
      numerador = 1
      denominador = 1
      other_pontos = np.delete(pontos, i, 0)

      for other_p in other_pontos:
         numerador = numerador * (x - other_p[0])
         denominador = denominador * (p[0] - other_p[0])

      L[i] = numerador / denominador
      i = i+1

   #reduzir os problemas com floats
   P = horner(L.multiply(pontos[..., 1])[0])
   Y = None

   if valor != None :
      Y = lambdify(x, P, 'numpy')
      Y = Y(valor)


   return P,Y

def main (argv):

   x =np.array([[1,0],[3,6],[4,24],[5,60]])
   P, Y = lagrange(x,3.5)
   print "\npolinômio de interpolação: ",
   print P
   print "f(3.5) = %f\n"%Y


if __name__ == '__main__':
   main(sys.argv[1:])
```

```
# -*- coding: latin-1 -*-
"""
* Resolução do exercício 10.2 (Neide Maria B. Franco. Cálculo Numérico. Pearson. 1ª Edição.)
*
* Executado como : franco_10.2.py
*
* Parâmetros usados para teste:
*                       python franco_10.2.py
*
"""

import sys
import math
from sympy import *
from sympy.matrices import *
import numpy as np

def lagrange(pontos, valor=None):
    x = Symbol("x")
    L = zeros(1, pontos.shape[0])
    i = 0

    for p in pontos:
        numerador = 1
        denominador = 1
        other_pontos = np.delete(pontos, i, 0)

        for other_p in other_pontos:
            numerador = numerador * (x - other_p[0])
            denominador = denominador * (p[0] - other_p[0])

        L[i] = numerador / denominador
        i = i+1

    #reduzir os problemas com floats
    P = horner(L.multiply(pontos[..., 1])[0])
    Y = None

    if valor != None :
        Y = lambdify(x, P, 'numpy')
        Y = Y(valor)


    return P,Y

def main (argv):

    x =np.array([[0,0],[1/6.,0.5],[1/2.,1]])
    P, Y = lagrange(x,None)
    print "\npolinômio de interpolação: ",
    print P
```

```python
if __name__ == '__main__':
    main(sys.argv[1:])
```

```python
# -*- coding: latin-1 -*-
"""
* Resolução do exercício 10.7 (Neide Maria B. Franco. Cálculo Numérico. Pearson. 1ª Edição.)
*
* Executado como : franco_10.7.py
*
* Parâmetros usados para teste:
*                        python franco_10.7.py
*
"""

import sys
import math
from sympy import *
from sympy.matrices import *
import numpy as np

def f(x):
    return 7*x**5 - 3*x**2 - 1
def df(x):
    return x*(35*x**3 - 6)
def ddf(x):
    return 140*x**3 - 6
def dddf(x):
    return 420*x**2
def ddddf(x):
    return 840*x

def r3(x,x0,x1,x2,x3):
    derv = []

    derv.append(abs(ddddf(x0)))
    derv.append(abs(ddddf(x1)))
    derv.append(abs(ddddf(x2)))
    derv.append(abs(ddddf(x3)))


    maximo = max(derv)

    aux = ((abs(x - x0 ) * abs(x - x1) * abs(x-x2)*abs(x - x3))/24.) *maximo


    return abs(aux)

def lagrange(pontos, valor=None):
    x = Symbol("x")
```

```
    L = zeros(1, pontos.shape[0])
    i = 0

    for p in pontos:
        numerador = 1
        denominador = 1
        other_pontos = np.delete(pontos, i, 0)

        for other_p in other_pontos:
            numerador = numerador * (x - other_p[0])
            denominador = denominador * (p[0] - other_p[0])

        L[i] = numerador / denominador
        i = i+1

    #reduzir os problemas com floats
    P = horner(L.multiply(pontos[..., 1])[0])
    Y = None

    if valor != None :
        Y = lambdify(x, P, 'numpy')
        Y = Y(valor)


    return P,Y

def main (argv):

    x = np.array([-3,-2,-1,0,1,2,3])
    res = f(x)
    print "\t\tx   | ",
    for i in range(len(x)):
        print "\t %d"%x[i],
    print"  |"
    print"\t\t----|--------------------------------------------------------"
    print "\t\tf(x)|",
    for i in range(len(res)):
        print"\t%d"%res[i],
    print"|"

    pontos = np.array([[-2,-237],[-1,-11],[0,-1],[1,3]])
    P,Y = lagrange(pontos)
    print "Lagrange = ",
    print P

    aux = r3(-0.5,-3,-2,-1,0)
    print "R3 = " ,
    print aux

if __name__ == '__main__':
    main(sys.argv[1:])
```

```
# -*- coding: latin-1 -*-
"""
* Resolução do exercício 10.7 (Neide Maria B. Franco. Cálculo Numérico. Pearson. 1ª Edição.)
*
* Executado como : franco_10.7.py
*
* Parâmetros usados para teste:
*                       python franco_10.7.py
*
"""

import sys
import math
from sympy import *
from sympy.matrices import *
import numpy as np

def f(x):
   return 7*x**5 - 3*x**2 - 1
def df(x):
   return x*(35*x**3 - 6)
def ddf(x):
   return 140*x**3 - 6
def dddf(x):
   return 420*x**2
def ddddf(x):
   return 840*x

def r3(x,x0,x1,x2,x3):
   derv = []

   derv.append(abs(ddddf(x0)))
   derv.append(abs(ddddf(x1)))
   derv.append(abs(ddddf(x2)))
   derv.append(abs(ddddf(x3)))


   maximo = max(derv)

   aux = ((abs(x - x0 ) * abs(x - x1) * abs(x-x2)*abs(x - x3))/24.) *maximo


   return abs(aux)

def lagrange(pontos, valor=None):
   x = Symbol("x")
   L = zeros(1, pontos.shape[0])
   i = 0

   for p in pontos:
      numerador = 1
      denominador = 1
```

```
        other_pontos = np.delete(pontos, i, 0)

        for other_p in other_pontos:
            numerador = numerador * (x - other_p[0])
            denominador = denominador * (p[0] - other_p[0])

        L[i] = numerador / denominador
        i = i+1

    #reduzir os problemas com floats
    P = horner(L.multiply(pontos[..., 1])[0])
    Y = None

    if valor != None :
        Y = lambdify(x, P, 'numpy')
        Y = Y(valor)


    return P,Y

def main (argv):

    x = np.array([-3,-2,-1,0,1,2,3])
    res = f(x)
    print "\t\tx   | ",
    for i in range(len(x)):
        print "\t %d"%x[i],
    print"  |"
    print"\t\t----|--------------------------------------------------------"
    print "\t\tf(x)|",
    for i in range(len(res)):
        print"\t%d"%res[i],
    print"|"

    pontos = np.array([[-2,-237],[-1,-11],[0,-1],[1,3]])
    P,Y = lagrange(pontos)
    print "Lagrange = ",
    print P

    aux = r3(-0.5,-3,-2,-1,0)
    print "R3 = " ,
    print aux

if __name__ == '__main__':
    main(sys.argv[1:])
```

```
# -*- coding: latin-1 -*-
"""
* Resolução do exercício 10.3 (Neide Maria B. Franco. Cálculo Numérico. Pearson. 1ª Edição.)
*
* Executado como : franco_10.3.py
*
```

```
* Parâmetros usados para teste:
*                               python franco_10.3.py
*
"""

import sys
import math
from sympy import *
from sympy.matrices import *
import numpy as np

def lagrange(pontos, valor=None):
    x = Symbol("x")
    L = zeros(1, pontos.shape[0])
    i = 0

    for p in pontos:
        numerador = 1
        denominador = 1
        other_pontos = np.delete(pontos, i, 0)

        for other_p in other_pontos:
            numerador = numerador * (x - other_p[0])
            denominador = denominador * (p[0] - other_p[0])

        L[i] = numerador / denominador
        i = i+1

    #reduzir os problemas com floats
    P = horner(L.multiply(pontos[..., 1])[0])
    Y = None

    if valor != None :
        Y = lambdify(x, P, 'numpy')
        Y = Y(valor)


    return P,Y

def main (argv):

    x =np.array([[1,1.5708],[2,1.5719],[3,1.5739]])
    P, Y = lagrange(x,2.5)
    print "\npolinômio de interpolação: ",
    print P
    print "K(2.5) = %f\n"%Y


if __name__ == '__main__':
    main(sys.argv[1:])
```

```
# -*- coding: latin-1 -*-
"""
* Resolução do exercício 10.4 (Neide Maria B. Franco. Cálculo Numérico. Pearson. 1ª Edição.)
*
* Executado como : franco_10-4.py
*
* Parâmetros usados para teste:
*                         python franco_10-4.py
*
"""

import sys
import math
from sympy import *
from sympy.matrices import *
import numpy as np

def lagrange(pontos, valor=None):
    x = Symbol("x")
    L = zeros(1, pontos.shape[0])
    i = 0

    for p in pontos:
        numerador = 1
        denominador = 1
        other_pontos = np.delete(pontos, i, 0)

        for other_p in other_pontos:
            numerador = numerador * (x - other_p[0])
            denominador = denominador * (p[0] - other_p[0])

        L[i] = numerador / denominador
        i = i+1

    #reduzir os problemas com floats
    P = horner(L.multiply(pontos[..., 1])[0])
    Y = None

    if valor != None :
        Y = lambdify(x, P, 'numpy')
        Y = Y(valor)


    return P,Y

def main (argv):

    x =np.array([[2.8,16.44],[3.0,20.08],[3.2,24.53]])
    P, Y = lagrange(x,3.1)
    print "\npolinômio de interpolação: ",
    print P
    print "e^3.1 = %f\n"%Y
```

```
if __name__ == '__main__':
    main(sys.argv[1:])
```

```
# -*- coding: latin-1 -*-
"""
* Resolução do exercício 10.5 (Neide Maria B. Franco. Cálculo Numérico. Pearson. 1ª Edição.)
*
* Executado como : franco_10-5.py
*
* Parâmetros usados para teste:
*                python franco_10-5.py
*
"""

import sys
import math
from sympy import *
from sympy.matrices import *
import numpy as np

def lagrange(pontos, valor=None):
    x = Symbol("x")
    L = zeros(1, pontos.shape[0])
    i = 0

    for p in pontos:
        numerador = 1
        denominador = 1
        other_pontos = np.delete(pontos, i, 0)

        for other_p in other_pontos:
            numerador = numerador * (x - other_p[0])
            denominador = denominador * (p[0] - other_p[0])

        L[i] = numerador / denominador
        i = i+1

    #reduzir os problemas com floats
    P = horner(L.multiply(pontos[..., 1])[0])
    Y = None

    if valor != None :
        Y = lambdify(x, P, 'numpy')
        Y = Y(valor)


    return P,Y

def main (argv):
```

```
    x =np.array([[0,-1],[0.5,-1.15],[1,0.63]])
    P, Y = lagrange(x,None)
    print "\npolinômio de interpolação: ",
    print P


if __name__ == '__main__':
    main(sys.argv[1:])
```