{NCM Theory Sheet}

John Doe
(King's College London)

Newton Method:

- Theory:

Take Taylor two terms of Taylor expansion:

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0)$$

As we approximate $f(x) = 0$, then we take

$$0 = f(x) \approx f'(x)(x - x_0)$$

This implies $x = x_0 - \frac{f(x_0)}{f'(x_0)}$

- Algorithm:

1. Start with initial guess: calculating $f(x_0)$ using given x_0
2. Find the tangent line using the derivative, i.e. $f(x) \approx f(x_0) + f'(x_0)(x - x_0)$
3. Find the x-intercept of the tangent line by $0 = f(x_0) + f'(x_0)(x - x_0)$, this value of x we call x_1
4. Repeat the process for x_1 by calculating $f(x_1)$

.....

-Code:

```
def f(x):
    return x**3+x**2
def df(x):
    return 3*x**2 + 2*x
def newton(x0, n):
    xs = [x0]
    for i in range(n):
        xs.append(xs[-1] - f(xs[-1])/df(xs[-1]))
```

Secant Method

- Theory

The idea is to use Secant method when derivative is difficult to compute. Using the linear approximation for the function $f(x)$ we get

$$f(x) \approx \frac{x - x_{n-1}}{x_{n-2} - x_{n-1}} f(x_{n-2}) + \\ + \frac{x - x_{n-2}}{x_{n-1} - x_{n-2}} f(x_{n-1}) \quad (1)$$

Applying the same method as for Newton:

$$\frac{x_n - x_{n-1}}{x_{n-2} - x_{n-1}} f(x_{n-2}) + \frac{x_n - x_{n-2}}{x_{n-1} - x_{n-2}} f(x_{n-1}) = 0$$

which gives:

$$x_n = \frac{x_{n-1} f(x_{n-2}) - x_{n-2} f(x_{n-1})}{f(x_{n-2}) - f(x_{n-1})}$$

It works the same way as Newton method in terms of algorithm

- Code:

```
def f(x):
    return x**3+x**2 #some comment
def df(x):
    return 3*x**2 + 2*x
def secant(x0, xm, n):
    x = [xm, x0]
    for i in range(n):
        xs.append(x[-1]*f(x[-2]) -
        x[-2]*f(xs[-1])/(f(x[-2]) - f(x[-1])))
```

Title

- Theory:

The idea is that at each step we divide the interval in two point and compute the midpoint between them as $c = \frac{a+b}{2}$

1. $f(c) = 0$, then we have found the zero $x^* = c$
2. $f(c)f(a) < 0$, then we can be sure that zero is between a and c and we repeat the procedure for the interval $[a, c]$
3. $f(c)f(a) > 0$ (which implies that $f(c)f(b) < 0$), and then zero is hidden between c and b. And we repeat the procedure again for $[c, b]$

- Code:

```
def f(x): \\
    return x**3+x**2\\

def bijection(a, b, n): \\
    for i in range(n): \\
        c=(a-b)/2 \\
        if abc(f(c))<1/10**10: \\
            print("found zero", c) \\
            return c
        if f(a)*f(c)<0:
            a=a
            b=c
        else:
            a=c
            b=b
        print("a=",a,"b=",b,"f(a)=",f(a))
```

{Python Cheat Sheet}

John Doe
(Universidade Federal do Triângulo Mineiro)

Outros Elementos

- Palavras-Chave

Oper.	Descrição
print	Imprime para a tela
while	"Enquanto- laço para repetição de alguma condição
for	"Para- loop para repetição de alguma condição
break	Interrompe o loop caso necessário
continue	Interrompe o loop atual sem sair do loop, reiniciando
if	"Se- usado para testar alguma condição
elif	É uma variante para o "senão- se a primeira condição falha, testa a próxima
else	"Senão- é opicional e será executado quando a primeira condição falhar
is	Testa a identidade do objeto
import	Importa outros módulos para dentro de um script
as	Usado para dar um apelido (alias) para um módulo
from	Para importar uma variável especifica, classe ou função de um módulo
def	Usado para criar uma função nova definida pelo usuário
return	Sai da função e retorna um valor
lambda	Cria uma função nova anônima
global	Acessa variáveis definida globalmente (fora de uma função)
try	Especifica manipuladores de exceções
except	Captura a exceção e executa códigos
finally	É sempre executado no final, utilizado para limpar os recursos
raise	Cria uma exceção definida pelo usuário
del	Deleta objetos
pass	Não faz nada
assert	Usado para fins de depuração
class	Usado para criar objetos definidos pelo usuário
exec	Executa dinamicamente um código Python
yield	É usado com geradores

Operadores Python

Tomemos como exemplo a=10 e b=20:

- Operadores Aritméticos

Op.	Descrição	Exemplo
+	Adição	a + b retorna: 30
-	Subtração	a - b retorna: -10
*	Multiplicação	a * b retorna: 200
/	Divisão	b / a retorna: 2
%	Módulo	a % b retorna: 0
**	Exponencial	a**b retorna: 10^{20}
//	Divisão Piso	9 // 2 retorna: 4

- Operadores de Comparação

As operações básicas de comparação podem ser usadas de diversas maneiras para todos os tipos de valores - números, strings, sequencias, listas, etc. O retorno será sempre True ou False.

Op.	Descrição	Exemplo
<	Menor que	a < b retorna: True
<=	Menor ou igual	a <= b retorna: True
==	Igual	a == b retorna: False
>	Maior que	a > b retorna: False
>=	Maior ou igual	a >= b retorna: False
!=	Diferente	a != b retorna: True
<>	Diferente	a <> b retorna: True

- Operadores Lógicos

Os operadores lógicos **and** e **or** Também retornam um valor booleano quando usado em uma estrutura de decisão.

Op.	Descrição
and	Se o resultado de ambos operadores é verdadeiro, retorna: True
or	Se um dos resultados retorna verdadeiro, retorna: True
not	É utilizado para reverter o estado lógico de qualquer operação booleana.

- Tuplas no Python

Tupla é uma lista de valores separados por vírgulas - é similar à uma lista porém é imutável:

```
uma_tupla = 'a','b','c','d','e'
outra_tupla = ('a','b','c','d','e')
```

- Números Aleatórios

Strings são compostos de caracteres:

```
uma_string = "Hello World!"
outra_string = 'Ola Mundo!"
```

Strings no Python

string é uma sequencia de caracteres geralmente usada para armazenar texto.
Strings são compostos de caracteres (não podem ser alterados - são imutáveis)

Criação

```
uma_string = "Hello World!"        outra_string = 'Ola Mundo!"
```

Acessando

```
uma_string[4]                      retorna: 'o'
```

(este caso retorna a 4ª posição do texto - começando a contar a partir do zero)

Dividindo

```
uma_string.split('')               retorna ['Hello','World']
```

(este caso divide o texto no espaço em branco em uma lista de duas strings)

```
uma_string.split('r')              retorna ['Hello Wo','ld']
```

(este caso divide o texto na letra 'r' em uma lista de duas strings)

Unindo

Para unir uma lista de strings usaremos a função join()

```
uma_lista = ["isto","eh","uma","lista","de","strings"]
' '.join(uma_lista)                retorna: "isto eh uma lista de strings"
' 'TESTE'.join(uma_lista)          Retorna:
''.join(uma_lista)                 retorna: "istoehumalistadestrings"
```

Formatando Strings

Podemos usar o operador % para adicionar elementos em uma string:

```
esta_string = "todos"
print("Olá para %s!"%esta_string)    retorna: "Olá para todos!"
```

- Operações com Strings

Definindo as variaveis de string para exemplo da seguinte forma: a = ['Hello'] e b = ['Python']

Oper.	Descrição	Exemplo
+	Concatenation - soma o conteúdo das duas strings	a + b retorna: HelloPython
*	Repetition - repete o conteúdo da string N vezes	a*2 retorna: HelloHello
.[ ]	Slice - fatia retornando o caractere no respectivo indice	a[1] retorna: "e"
.[:]	Range Slice - retorna os caracteres do intervalo indicado	a[1:4] retorna: "ell"
in	Membership - se o caractere existe na string, retorna true	H in a will give 1
not in	Membership - se o caractere não existe na string, retorna true	M not in a retorna: 1
%	Format - formata uma string	exemplos na tabela seguinte

- Formatação de Strings

Símbolo	Conversão	Símbolo	Conversão
%c	caractere	%i	decimal inteiro com sinal
%d	decimal inteiro com sinal	%u	decimal inteiro sem sinal
%o	octal inteiro	%x	hexadecimal inteiro (letras minúsculas)
%f	numero real ponto flutuante	%X	hexadecimal inteiro (letras maiúsculas)
%g	o menor entre %f e %e	%e	notação exponencial (com 'e' minúsculo)
%G	o menor entre %f e %E	%E	notação exponencial (com 'E' maiúsculo)
.	.	%s	converção de string via str() antes de formatar