PROJETO MÁRIO TRAVASSOS

Artigo Científico

MÉTODO DE NEWTON E RELAÇÕES DE GIRARD: COMO CONECTAR O CÁLCULO, A FÍSICA E A COMPUTAÇÃO NA FORMAÇÃO ACADÊMICA OFERECIDA PELA EsPCEx.

Cap Carlos Eduardo Guedes BELCHIOR

Cap R1 VALDECI Otacilio dos Santos

1º Ten Felipe QUIRINO Andre

(opinião de inteira responsabilidade dos autores)

2022

ESCOLA PREPARATÓRIA DE CADETES DO EXÉRCITO

DIVISÃO DE ENSINO

PROJETO MÁRIO TRAVASSOS

MÉTODO DE NEWTON E RELAÇÕES DE GIRARD: COMO CONECTAR O CÁLCULO, A FÍSICA E A COMPUTAÇÃO NA FORMAÇÃO ACADÊMICA OFERECIDA PELA EsPCEx.

Carlos Eduardo Guedes Belchior[1]

Valdeci Otacilio dos Santos[2]

Felipe Quirino Andre[3]

CAMPINAS

2022

---

[1] Mestre em Matemática pelo Instituto de Matemática Pura e Aplicada. Especialista em Ciências Militares pela EsAO. Especialista em Aplicações Complementares às Ciências Militares pela EsAEx (atual EsFCEx). Licenciado e bacharel em Matemática pela Universidade Federal de São Carlos.

E-mail: belchior.mat@gmail.com

[2] Mestre em Engenharia Elétrica pela Universidade Estadual de Campinas. Especialista em Gestão da Segurança da Informação e Comunicações pela Universidade de Brasília. Graduado em Tecnologia em Processamento de Dados pela Faculdade de Tecnologia de Americana.

E-mail: valdeciodsc@gmail.com

[3] Mestre em Matemática pela Universidade Federal Rural do Rio de Janeiro. Especialista em Aplicações Complementares às Ciências Militares pela EsFCEx. Licenciado em Matemática pela Universidade Federal Fluminense.

E-mail: f.quirino@hotmail.com

SUMÁRIO

1 INTRODUÇÃO

1.1 Revisão de literatura

A interdisciplinaridade, entendida sob o conceito de estabelecer relações entre duas ou mais disciplinas ou ramos de conhecimento de acordo com o dicionário Houaiss da Língua Portuguesa, é uma das características que deve ser buscada pelo docente atual com a finalidade de fornecer ferramentas para enriquecer a visão de mundo dos alunos (FIA BUSINESS SCHOOL, 2021).

Em consonância com os Parâmetros Curriculares Nacionais (BRASIL, 1999), é essencial o trabalho interdisciplinar, contudo a interdisciplinaridade não tem a pretensão de criar novas disciplinas ou saberes, mas de utilizar os conhecimentos de várias disciplinas para resolver um problema concreto ou compreender um determinado fenômeno sob diferentes pontos de vista.

No século XXI, convive-se com a evolução das Tecnologias da Informação e Comunicação (TIC). Isso fez com que, rapidamente, diversas áreas do conhecimento sofressem mudanças cada vez mais significativas. Em consequência, os modelos de educação e sociedade necessitavam de inovação (NOJOSA; LIMA; RIBEIRO, 2018).

Nesse sentido, é razoável que disciplinas das ciências exatas, em particular, o Cálculo e a Física, interajam com a Computação, os conhecimentos e os instrumentos visando, nos dias atuais, à otimização do ensino.

1.2 Objetivo

O presente trabalho intenta oferecer ferramentas para que alguns objetos de conhecimento das disciplinas de Cálculo, Física e Computação do Curso de Formação e Graduação de Oficiais de Carreira da Linha de Ensino Militar Bélico (CFO/LEMB) sejam interligados de modo que tenham mais significado para o corpo discente da Escola Preparatória de Cadetes do Exército (EsPCEx), haja vista que “podem ser ensinadas determinadas técnicas que tornem esse processo (o de aprender a resolver, resolver para aprender) mais eficiente” (POZO, 1998). Concretamente, o objetivo consiste em conectar o Cálculo, a Física e a Computação na formação acadêmica oferecida pela EsPCEx por meio do Método de Newton e das Relações de Girard.

2 DESENVOLVIMENTO

2.1 Motivação pedagógica

Desde o Ensino Fundamental, os alunos aprendem uma forma particular das Relações de Girard, qual seja, o cálculo da soma e do produto, por meio dos coeficientes do polinômio quadrático, das raízes de uma equação do segundo grau.

Uma vez inseridos no contexto do ensino superior, devem assimilar – e aplicar a uma gama de problemas físicos – conceitos do Cálculo Diferencial e Integral, como, por exemplo, a obtenção da função horária da velocidade V como taxa de variação da posição S em relação ao tempo, o que pode ser expresso em termos da derivada: $V(t) = S'(t)$.

Ora, se S for uma função polinomial do terceiro grau, então o problema de classificar o movimento em acelerado ou retardado, bem como em progressivo ou retrógrado, passa pela obtenção das soluções de $V(t) = 0$, ou seja, as raízes de uma equação do segundo grau, que, neste caso, serão os valores candidatos aos tempos nos quais, eventualmente, há inversão do sentido do movimento.

As Relações de Girard facilitam a obtenção de tais raízes desde que sejam inteiras, mas também se forem fracionárias – com uma técnica que será apresentada na próxima seção. Quando isso não for possível, bastará obter valores aproximados e, com esse fim, o Método de Newton se mostra uma ferramenta poderosíssima, conforme se verá adiante.

Por fim, pode-se lançar mão de algoritmos computacionais para implementar de modo eficiente quaisquer das técnicas supracitadas, e esse será o objeto de discussão do último tópico tratado no presente artigo.

2.2 Situação-problema

Considere um móvel cuja posição S (em metros), em função do tempo $t \geq 0$ (em segundos), seja dada por

$$S(t) = 2t^3/3 - 5t^2/2 + 2t + 5/3$$

Em qual posição estará esse móvel quando ocorrer a inversão do seu sentido no maior tempo possível?

Quando há inversão do sentido, a velocidade do móvel é nula, logo os instantes de tempo nos quais isso *pode* ocorrer são as soluções da equação

$$2t^2 - 5t + 2 = 0 ,$$

cujos produto P e soma S das raízes são dados pelas Relações de Girard abaixo:

$$P = c/a = 2/2 = 1$$

$$S = -b/a = 5/2$$

Como se vê, as raízes de tal equação quadrática não podem ser ambas inteiras… mas um dos objetivos deste artigo é justamente adaptar as Relações Girard para obter as frações correspondentes às raízes, de acordo com o que segue.

Se R1 e R2 são as raízes da equação acima, então pondo N1 = a.R1 e N2 = a.R2, tem-se

$$N1 + N2 = a.(R1 + R2) = a.S = -b$$

$$N1.N2 = a.R1.a.R2 = a.a.P = a.c$$

logo

$$N1 + N2 = 5$$

$$N1.N2 = 4$$

e, dessa forma, N1 = 1 e N2 = 4, de modo que R1 = ½ e R2 = 4/2 = 2.

Na situação apresentada, foi possível obter com exatidão as raízes, mas como isso nem sempre é viável pela técnica usada, veremos agora como empregar um processo recursivo para aproximá-las. Mais precisamente, vamos abordar o Método de Newton.

A ideia é partir de um ponto arbitrário (x1,y1) do gráfico da função f (se y1=0, já se tem x1 como uma das raízes), traçar a reta tangente ao gráfico em (x1,y1) e calcular x2 a abscissa do ponto onde essa reta intersecta o eixo Ox (isso só não será possível se a reta for horizontal, ou seja, caso f'(x1) = 0). Em seguida, repete-se o processo para (x2,y2), onde y2 = f(x2). Após um número conveniente de iterações, obtém-se uma boa aproximação x para alguma das raízes de f.

Por exemplo, para

$$f(x) = 2x^2 - 5x + 2,$$

f'(x) = 4x – 5 e, começando com x1 = 0, tem-se y1 = 2, f'(x1) = –5 e a primeira reta tangente **g** tem equação dada por

$$\mathbf{g}: y = 2 - 5.x$$

cuja intersecção com o eixo das abscissas se dá em x2 = ⅖. Dessa forma, y2 = 8/25 e f'(x2) = –17/5, de modo que a segunda reta tangente **i** é dada pela equação

$$\mathbf{i}: y = 8/25 - 17/5(x-2/5) = 42/25 - (17/5).x$$

cuja intersecção com o eixo das abscissas se dá em x3 = 42/85 ≅ 0,49, o que difere menos de um centésimo de um dos zeros de f: R1 = ½.

Se, porém, o "chute inicial" fosse x1 = 3, a sequência de procedimentos acima forneceria x2 ≅ 2,29 e x3 ≅ 2,04, ou seja, aproximam-se do outro zero de f: R2 = 2.

A figura a seguir ilustra geometricamente o procedimento ora descrito:

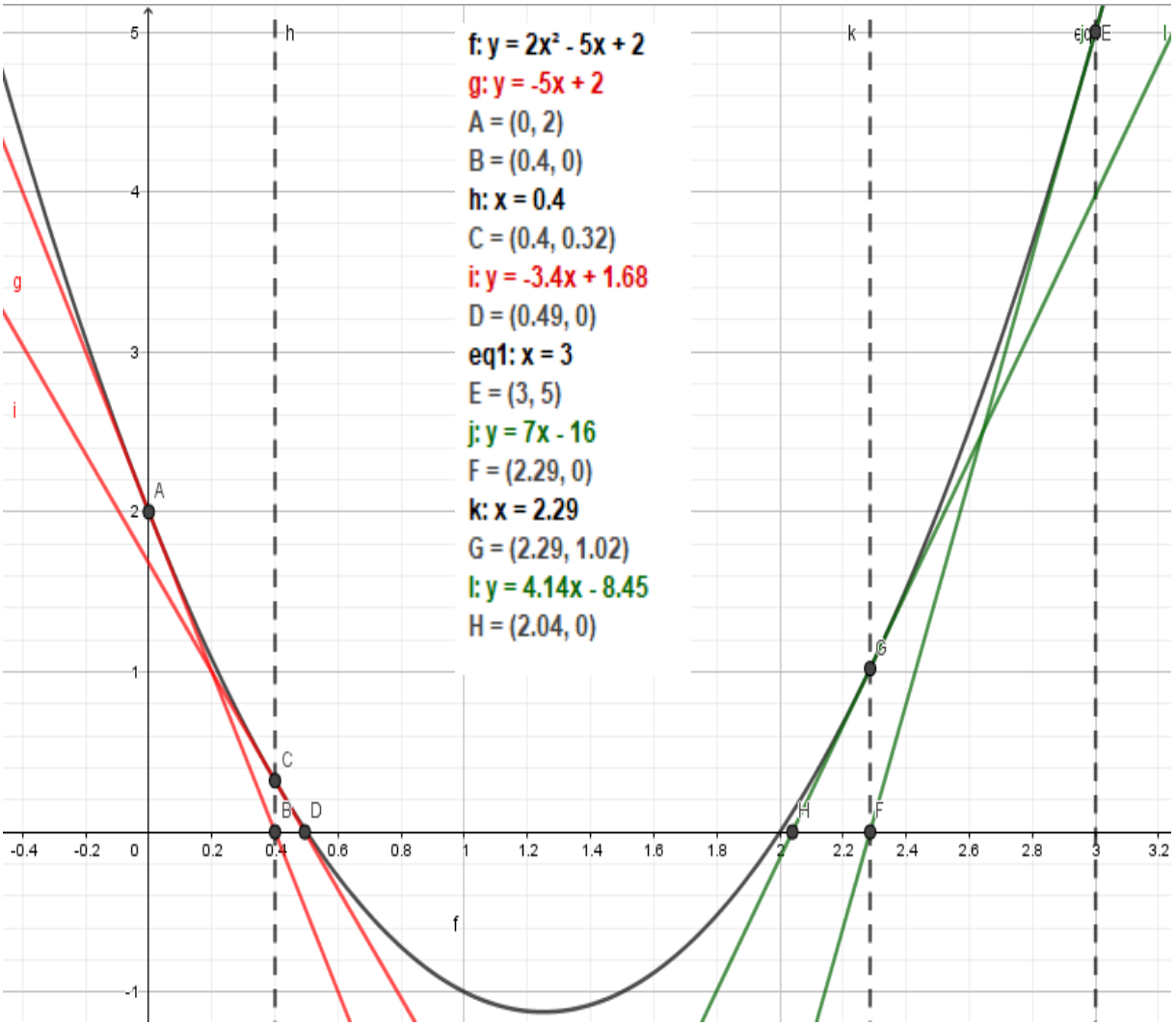

Figura I: Gráfico de f e das retas tangentes auxiliares para obtenção de xN.

Na Figura I, tem-se a seguinte correspondência com os pontos obtidos analiticamente: A(x1,y1); C(x2,y2) e x3 é a abscissa de D no primeiro processo iterativo. Já no segundo, E(x1,y1); G(x2,y2) e x3 é a abscissa de H.

2.3 Implementação computacional

De modo geral, o Método de Newton pode ser formulado pela seguinte recursão:

$$x\{K+1\} = xK - yK/f'(xK) \qquad (\#)$$

que pode ser implementada por meio de um algoritmo computacional, conforme apresentado a seguir, ou, até mesmo, fazendo uso de qualquer planilha eletrônica.

A Figura II apresenta um algoritmo que implementa o Método de Newton, codificado na linguagem de programação Python.

O algoritmo mostrado na Figura II inicia-se com a definição de duas sub-rotinas (funções). A primeira função (linhas 1 a 13) tem o objetivo de receber e validar os coeficientes do polinômio p(x), retornando tais valores para o programa principal. Já a segunda (linhas 15 a 21) calcula, imprime e retorna ao programa principal os coeficientes da função derivada de p(x).

O programa principal (linhas 23 a 52) tem início com a invocação das funções descritas no parágrafo anterior (linhas 24 a 31) e, em seguida, solicita ao usuário que insira um valor inicial para x (linha 32). Nas linhas 33 a 40, o algoritmo trata os casos particulares em que o valor de x, digitado, constitui um vértice da parábola (linhas 34 a 36) e/ou corresponde uma das raízes do polinômio. O algoritmo permite que o usuário especifique o número de iterações desejadas (linha 43) e finaliza calculando e imprimindo os valores de x para cada iteração (linhas 44 a 52).

```python
#Definição da função que recebe os valores dos coeficientes:
def rec_coef():
    print("\nDIGITE OS VALORES DOS COEFICIENTES DO POLINÔMIO QUADRÁTICO p(x)")
    r_a = float(input("Digite o valor do coeficiente \"a\": "))
    r_b = float(input("Digite o valor do coeficiente \"b\": "))
    r_c = float(input("Digite o valor do coeficiente \"c\": "))
    while r_b**2-4*r_a*r_c<0:
        print("\nEsse polinômio não possui raízes reais!")
        print("Escolha novos coeficientes!")
        r_a = float(input("Digite o valor do coeficiente \"a\": "))
        r_b = float(input("Digite o valor do coeficiente \"b\": "))
        r_c = float(input("Digite o valor do coeficiente \"c\": "))
    return r_a,r_b,r_c

#Definição da função que calcula e imprime os coeficientes da derivada:
def c_coef_der(a_der,b_der):
    print("\nCoeficientes da função derivada de p(x):")
    a_der = a_der*2
    print("Coeficiente \"a\" da função derivada:",a_der)
    print("Coeficiente \"b\" da função derivada:",b_der)
    return a_der,b_der

#Programa principal:
coef = rec_coef()
a = coef[0]
b = coef[1]
c = coef[2]
coef_der = c_coef_der(a,b)
a_d = coef_der[0]
b_d = coef_der[1]
encerra = False
ini = float(input("Digite o valor incial de x: "))
while (2*a*ini+b==0)or(a*ini**2+b*ini+c==0):
    if 2*a*ini+b==0:
        print("\nEsse \"x\" corresponde a um vértice da parábola!")
        ini = float(input("Digite um novo valor incial de x: "))
    else:
        print("\nEsse valor de \"x\" corresponde a uma das raízes do polinômio!")
        encerra = True
        break
if encerra == False:
    valores = [ini]
    iter = int(input("Digite o número de iterações desejadas: "))
    for i in range(iter):
        x = valores[i]-(((a*valores[i]**2)+(b*valores[i])+c)/(a_d*valores[i]+b_d))
        valores.append(x)
    print("\nVetor de valores de \"x\" obtidos:")
    print(valores)
    print("\nIMPRESSÃO DOS VALORES OBTIDOS EM CADA ITERAÇÃO:")
    print("Valor inicial de x:",valores[0])
    for j in range(1,len(valores)):
        print("Valor de x após a "+str(j)+"ª iteração: ",valores[j])
```

Figura II: Método de Newton codificado na linguagem Python.

A Figura III apresenta o resultado da execução do algoritmo acima, para o polinômio $2x^2 – 5x + 2$, em que foram escolhidos um valor inicial de x igual a zero e cinco iterações.

```
DIGITE OS VALORES DOS COEFICIENTES DO POLINÔMIO QUADRÁTICO p(x)
Digite o valor do coeficiente "a": 2
Digite o valor do coeficiente "b": -5
Digite o valor do coeficiente "c": 2

Coeficientes da função derivada de p(x):
Coeficiente "a" da função derivada: 4.0
Coeficiente "b" da função derivada: -5.0

Digite o valor incial de x: 0
Digite o número de iterações desejadas: 5

Vetor de valores de "x" obtidos:
[0.0, 0.4, 0.4941176470588235, 0.49997711146715496, 0.49999999965075403, 0.5]

IMPRESSÃO DOS VALORES OBTIDOS EM CADA ITERAÇÃO:
Valor inicial de x: 0.0
Valor de x após a 1ª iteração:  0.4
Valor de x após a 2ª iteração:  0.4941176470588235
Valor de x após a 3ª iteração:  0.49997711146715496
Valor de x após a 4ª iteração:  0.49999999965075403
Valor de x após a 5ª iteração:  0.5
```

Figura III: Algoritmo que implementa o Método de Newton, executado.

Se desejar implementar a recursão do Método de Newton, por meio de uma planilha eletrônica, pode-se proceder conforme ilustrado na Figura IV.

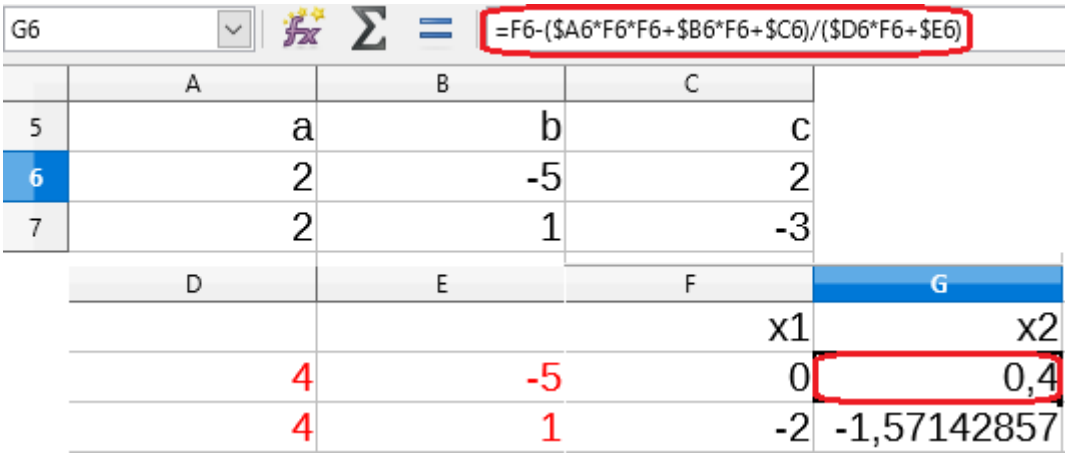

Figura IV: Método de Newton em planilha eletrônica.

As entradas a, b e c correspondem aos coeficientes do polinômio quadrático p(x) e, nas colunas D e E, estão os da sua derivada p'(x); x1 é o "chute" inicial que o usuário insere para gerar as aproximações xN, N>1.

A Linha 6 tem os coeficientes do polinômio $2x^2 - 5x + 2$ e de sua derivada, **4x – 5**. A célula **G6** traz a fórmula (#) em função de tais coeficientes e do valor inicial **x1 = 0**:

F6-($A6***F6*****F6**+$B6***F6**+$C6)/($D6***F6**+$E6)

A Linha 7 tem os coeficientes do polinômio $2x^2 + x - 3$, e de sua derivada **4x + 1**, para o qual foi escolhido o valor inicial **x1 = –2**.

Para aplicar a recorrência, basta deslizar a célula **G6** para a direita e, assim, aproximar uma das raízes. No primeiro polinômio, a sequência (xN) tende para ½ e, no segundo, para –3/2, conforme se vê a seguir.

A	B	C	D	E
a	b	c		
2	-5	2	4	-5
2	1	-3	4	1

F	G	H	I	J
x1	x2	x3	x4	x5
0	0,4	0,494117647	0,499977111	0,5
-2	-1,57142857	-1,5019305	-1,50000149	-1,5

Figura V: Sequência gerada pelo Método de Newton (4 iterações) a partir dos valores iniciais 0 e –2.

Se, por outro lado, fossem tomados como valores iniciais, respectivamente, 3 e 0, as sequências correspondentes tenderiam a 2 e 1, como mostrado abaixo:

F	G	H	I	J
x1	x2	x3	x4	x5
3	2,285714286	2,039408867	2,000983685	2,000000644
0	3	1,615384615	1,101506741	1,0038119

Figura VI: Sequência gerada pelo Método de Newton (4 iterações) a partir dos valores iniciais 3 e 0.

Observação: i) Se o discriminante ($b^2 - 4.a.c$) for negativo, o polinômio p(x) não admitirá raízes reais, logo a sequência (xN) não convergirá; **ii)** Caso o valor inicial x1 corresponda a um vértice, não será possível executar o Método de Newton, visto que p'(x1)=0, de modo que o programa deverá solicitar ao usuário que escolha um novo valor inicial.

Por fim, para implementar as Relações de Girard (adaptadas para raízes fracionárias), devem-se seguir estes passos:

i) O usuário entra com os coeficientes a, b e c;

ii) O programa determina os inteiros positivos (até $|a.c|^{1/2}$) D que são divisores de a.c;

iii) Verifica-se, dentre tais divisores, se –b é igual a D + a.c/D (ou –D – a.c/D);

iv) A saída é (R1,R2) = (D/a , c/D) (ou (–D/a , –c/D)), onde D satisfaz à condição iii.

No caso do polinômio $2x^2 + x – 3$, por exemplo, o usuário insere os números

2 1 –3

ii) 1 e 2

iii) 1 + (–6)/1 = –5; **2 + (–6)/2 = –1**; –1 + (–6)/(–1) = 5; –2 + (–6)/(–2) = 1

Por fim, o programa fornece (R1,R2) = (1 ; –1,5).

Um algoritmo que implementa as Relações de Girard (adaptadas para raízes fracionárias), codificado na linguagem de programação Python, é apresentado na Figura VII.

```
1  #Definição da função que recebe os valores dos coeficientes:
2  def rec_coef():
3      print("\nDIGITE OS VALORES DOS COEFICIENTES DO POLINÔMIO QUADRÁTICO p(x)")
4      r_a = int(input("Digite o valor do coeficiente \"a\": "))
5      r_b = int(input("Digite o valor do coeficiente \"b\": "))
6      r_c = int(input("Digite o valor do coeficiente \"c\": "))
7      while r_b**2-4*r_a*r_c<0:
8          print("\nEsse polinômio não possui raízes reais!")
9          print("Escolha novos coeficientes!")
10         r_a = int(input("Digite o valor do coeficiente \"a\": "))
11         r_b = int(input("Digite o valor do coeficiente \"b\": "))
12         r_c = int(input("Digite o valor do coeficiente \"c\": "))
13     return r_a,r_b,r_c
14
15 #Programa principal:
16 coef = rec_coef()
17 a = coef[0]
18 b = coef[1]
19 c = coef[2]
20 lista =[]
21 lm = int((abs(a*c))**(1/2))
22 for n in range(1,lm+1):
23     if lm%n == 0:
24         if n + (a*c)/n == -b:
25             l = n
26             m = int((a*c)/n)
27         if -n - (a*c)/n == -b:
28             l = -n
29             m = int(-(a*c)/n)
30 lista.append(l/a)
31 lista.append(m/a)
32 r1r2 = tuple(lista)
33 if r1r2[0] != r1r2[1]:
34     print("\nValores das raízes exatas: ",r1r2)
35 else:
36     print("\nO valor",r1r2[0],"é a raiz dupla desse polinômio!")
```

Figura VII: Relações de Girard codificadas na linguagem Python.

Da mesma forma que ocorreu com o algoritmo que implementa o Método de Newton, o algoritmo apresentado na Figura VII inicia-se com a definição de uma função, que recebe e valida os coeficientes do polinômio p(x), retornando tais valores para o programa principal (linhas 1 a 13).

O programa principal (linhas 15 a 36) realiza o cálculo das raízes exatas do polinômio (linhas 15 a 32), imprimindo o resultado do processamento (linhas 33 a 36).

A Figura VIII apresenta o algoritmo que implementa as Relações de Girard sendo executado, para o polinômio $2x^2 + x - 3$.

```
DIGITE OS VALORES DOS COEFICIENTES DO POLINÔMIO QUADRÁTICO p(x)
Digite o valor do coeficiente "a": 2
Digite o valor do coeficiente "b": 1
Digite o valor do coeficiente "c": -3

Valores das raízes exatas:  (1.0, -1.5)
```

Figura VIII: Algoritmo que implementa as Relações de Girard, executado.

O apêndice a este artigo contempla um algoritmo, codificado na linguagem de programação Python, que implementa o Método de Newton e as Relações de Girard em um mesmo programa. Tal algoritmo possibilita a entrada dos coeficientes do polinômio e permite que o usuário insira um valor inicial de"x" quantas vezes desejar, além da escolha do número de iterações a ser realizada. O algoritmo permite, também, que o usuário, ao decidir não continuar inserindo novos valores para "x", escolha se deseja obter as raízes exatas do polinômio em questão e, caso sua resposta seja afirmativa, tais resultados serão mostrados ao usuário.

3 CONCLUSÃO

Este artigo expôs os fundamentos teóricos de um algoritmo, codificado na linguagem de programação Python, que implementa o Método de Newton e as Relações de Girard em um mesmo programa.

Concluiu-se, por fim, a existência e simplicidade de uma de conexão possível do Cálculo, da Física e da Computação na aprendizagem oferecida durante a formação acadêmica na EsPCEx.

O presente trabalho alvorece um vasto horizonte de conhecimento no qual se queiram interligar os conceitos matemáticos com modelos físicos e programas de computação.

REFERÊNCIAS

BRASIL. **Parâmetros Curriculares Nacionais para o Ensino Médio (PCNEM)**. Brasília, 1999. Disponível em
<Publicações - Ministério da Educação (mec.gov.br)>. Acesso em: 25 Agosto 2022.

CENTRO DE REFERÊNCIAS EM EDUCAÇÃO INTEGRAL. **Interdisciplinaridade.** 2013. Disponível em <Interdisciplinaridade - Centro de Referências em Educação Integral (educacaointegral.org.br)>. Acesso em: 24 Agosto 2022.

DULLIUS, M. M; HENNEMANN, N. R.; KÖHNLEIN, M. M.; MARCHI, M. I. **Fontes de energia:** uma proposta interdisciplinar no Ensino de Ciências Exatas. XII Salão de Iniciação Científica PUCRS, Porto Alegre, 2011. Disponível em <1.pdf (pucrs.br)>. Acesso em: 30 Agosto 2022.

FIA BUSINESS SCHOOL. **Interdisciplinaridade:** Conceito, importância e vantagens. Pinheiros, 2021. Disponível em <Interdisciplinaridade: Conceito, importância e vantagens - FIA>. Acesso em: 29 Agosto 2022.

NOJOSA, D.M.B.; LIMA, I.B.; RIBEIRO, J.W. **Interdisciplinaridade no ensino de Ciências e Matemática**. Imprensa Universitária da Universidade Federal do Ceará, 2018.

PEDRO DA SILVA, M. N. **Relações de Girard nas equações do 3º e do 4º grau.** Mundo Educação. Disponível em
<Relações de Girard nas equações do 3º e do 4º grau - Mundo Educação (uol.com.br)>. Acesso em: 25 Agosto 2022.

POZO, J. I. **A solução de problemas**: aprender a resolver, resolver para aprender. Porto Alegre: ArtMed, p.13-29, 1998.

APÊNDICE

CÓDIGO-FONTE DO PROGRAMA

```
#Definição da função que calcula as raízes exatas:
def calcula_raizes(coef_a,coef_b,coef_c):
    lista =[]
    lm = int((abs(a*c))**(1/2))
    for n in range(1,lm+1):
        if lm%n == 0:
            if n + (a*c)/n == -b:
                l = n
                m = int((a*c)/n)
            if -n - (a*c)/n == -b:
                l = -n
                m = int(-(a*c)/n)
    lista.append(l/a)
    lista.append(m/a)
    r1r2 = tuple(lista)
    print("Valores das raízes exatas: ",r1r2)

#Definição da função que recebe os valores dos coeficientes:
def rec_coef():
    print("\nDIGITE OS VALORES DOS COEFICIENTES DO POLINÔMIO QUADRÁTICO p(x)")
    r_a = float(input("Digite o valor do coeficiente \"a\": "))
    r_b = float(input("Digite o valor do coeficiente \"b\": "))
    r_c = float(input("Digite o valor do coeficiente \"c\": "))
    while r_b**2-4*r_a*r_c<0:
        print("\nEsse polinômio não possui raízes reais!")
        print("Escolha novos coeficientes!")
        r_a = float(input("Digite o valor do coeficiente \"a\": "))
        r_b = float(input("Digite o valor do coeficiente \"b\": "))
        r_c = float(input("Digite o valor do coeficiente \"c\": "))
    return r_a,r_b,r_c

#Definição da função que calcula e imprime os coeficientes da derivada:
def c_coef_der(a_der,b_der):
    print("\nCoeficientes da função derivada de p(x):")
    a_der = a_der*2
    print("Coeficiente \"a\" da função derivada:",a_der)
    print("Coeficiente \"b\" da função derivada:",b_der)
    return a_der,b_der

```

```
#Programa Principal:
coef = rec_coef()
a = coef[0]
b = coef[1]
c = coef[2]
coef_der = c_coef_der(a,b)
a_d = coef_der[0]
b_d = coef_der[1]
encerra = False
resp = "sim"
while resp == "sim":
    ini = float(input("Digite o valor inicial de x: "))
    while (2*a*ini+b==0)or(a*ini**2+b*ini+c==0):
        if 2*a*ini+b==0:
            print("\nEsse \"x\" corresponde a um vértice da parábola!")
            ini = float(input("Digite um novo valor incial de x: "))
        else:
            print("\nEsse valor de \"x\" corresponde a uma das raízes do polinômio!")
            encerra = True
            break
    if encerra == False:
        valores = [ini]
        iter = int(input("Digite o número de iterações desejadas: "))
        for i in range(iter):
            x = valores[i]-(((a*valores[i]**2)+(b*valores[i])+c)/(a_d*valores[i]+b_d))
            valores.append(x)
        print("\nVetor de valores de \"x\" obtidos:")
        print(valores)
        print("\nIMPRESSÃO DOS VALORES OBTIDOS EM CADA ITERAÇÃO:")
        print("Valor inicial de x:",valores[0])
        for j in range(1,len(valores)):
            print("Valor de x após a "+str(j)+"ª iteração: ",valores[j])
        resp = input("\nDESEJA FAZER UMA NOVA BUSCA? Digite SIM ou NÃO: ")
        resp = resp.lower()
if resp == "não":
    opcao = input("\nDeseja calcular as raizes exatas? Digite SIM ou NÃO: ")
    opcao = opcao.lower()
    if opcao == "sim":
        calcula_raizes(int(a),int(b),int(c))
print("\n***FINAL DO PROGRAMA***")
```

```python
#Definição da função que calcula as raizes exatas:
def calcula_raizes(coef_a,coef_b,coef_c):
    lista =[]
    lm = int((abs(a*c))**(1/2))
    for n in range(1,lm+1):
        if lm%n == 0:
            if n + (a*c)/n == -b:
                l = n
                m = int((a*c)/n)
            if -n - (a*c)/n == -b:
                l = -n
                m = int(-(a*c)/n)
    lista.append(l/a)
    lista.append(m/a)
    r1r2 = tuple(lista)
    print("Valores das raizes exatas: ",r1r2)

#Definição da função que recebe os valores dos coeficientes:
def rec_coef():
    print("\nDIGITE OS VALORES DOS COEFICIENTES DO POLINÔMIO QUADRÁTICO p(x)")
    r_a = float(input("Digite o valor do coeficiente \"a\": "))
    r_b = float(input("Digite o valor do coeficiente \"b\": "))
    r_c = float(input("Digite o valor do coeficiente \"c\": "))
    while r_b**2-4*r_a*r_c<0:
        print("\nEsse polinômio não possui raizes reais!")
        print("Escolha novos coeficientes!")
        r_a = float(input("Digite o valor do coeficiente \"a\": "))
        r_b = float(input("Digite o valor do coeficiente \"b\": "))
        r_c = float(input("Digite o valor do coeficiente \"c\": "))
    return r_a,r_b,r_c

#Definição da função que calcula e imprime os coeficientes da derivada:
def c_coef_der(a_der,b_der):
    print("\nCoeficientes da função derivada de p(x):")
    a_der = a_der*2
    print("Coeficiente \"a\" da função derivada:",a_der)
#Programa Principal:
coef = rec_coef()
a = coef[0]
b = coef[1]
c = coef[2]
coef_der = c_coef_der(a,b)
a_d = coef_der[0]
b_d = coef_der[1]
encerra = False
resp = "sim"
while resp == "sim":
    ini = float(input("Digite o valor inicial de x: "))
    while (2*a*ini+b==0)or(a*ini**2+b*ini+c==0):
        if 2*a*ini+b==0:
            print("\nEsse \"x\" corresponde a um vértice da parábola!")
            ini = float(input("Digite um novo valor incial de x: "))
        else:
            print("\nEsse valor de \"x\" corresponde a uma das raizes do polinômio!")
            encerra = True
            break
    if encerra == False:
        valores = [ini]
        iter = int(input("Digite o número de iterações desejadas: "))
        for i in range(iter):
            x = valores[i]-(((a*valores[i]**2)+(b*valores[i])+c)/(a_d*valores[i]+b_d))
            valores.append(x)
        print("\nVetor de valores de \"x\" obtidos:")
        print(valores)
        print("\nIMPRESSÃO DOS VALORES OBTIDOS EM CADA ITERAÇÃO:")
        print("Valor inicial de x:",valores[0])
        for j in range(1,len(valores)):
            print("Valor de x após a "+str(j)+"ª iteração: ",valores[j])
        resp = input("\nDESEJA FAZER UMA NOVA BUSCA? Digite SIM ou NÃO: ")
        resp = resp.lower()
if resp == "não":
    opcao = input("\nDeseja calcular as raizes exatas? Digite SIM ou NÃO: ")
    opcao = opcao.lower()
    if opcao == "sim":
        calcula_raizes(int(a),int(b),int(c))
print("\n***FINAL DO PROGRAMA***")
```

PROGRAMA EM EXECUTADO COM O POLINÔMIO $4x^2 - 17x + 4$

```
DIGITE OS VALORES DOS COEFICIENTES DO POLINÔMIO QUADRÁTICO p(x)
Digite o valor do coeficiente "a": 4
Digite o valor do coeficiente "b": -17
Digite o valor do coeficiente "c": 4

Coeficientes da função derivada de p(x):
Coeficiente "a" da função derivada: 8.0
Coeficiente "b" da função derivada: -17.0
Digite o valor inicial de x: 0
Digite o número de iterações desejadas: 4

Vetor de valores de "x" obtidos:
[0.0, 0.23529411764705882, 0.2499427786678874, 0.2499999991268851, 0.25]

IMPRESSÃO DOS VALORES OBTIDOS EM CADA ITERAÇÃO:
Valor inicial de x: 0.0
Valor de x após a 1ª iteração:  0.23529411764705882
Valor de x após a 2ª iteração:  0.2499427786678874
Valor de x após a 3ª iteração:  0.2499999991268851
Valor de x após a 4ª iteração:  0.25

DESEJA FAZER UMA NOVA BUSCA? Digite SIM ou NÃO: SIM
Digite o valor inicial de x: 5
Digite o número de iterações desejadas: 4

Vetor de valores de "x" obtidos:
[5.0, 4.173913043478261, 4.007380924922154, 4.000014470517626, 4.000000000055838]

IMPRESSÃO DOS VALORES OBTIDOS EM CADA ITERAÇÃO:
Valor inicial de x: 5.0
Valor de x após a 1ª iteração:  4.173913043478261
Valor de x após a 2ª iteração:  4.007380924922154
Valor de x após a 3ª iteração:  4.000014470517626
Valor de x após a 4ª iteração:  4.000000000055838

DESEJA FAZER UMA NOVA BUSCA? Digite SIM ou NÃO: NÃO

Deseja calcular as raizes exatas? Digite SIM ou NÃO: SIM
Valores das raízes exatas:  (0.25, 4.0)

***FINAL DO PROGRAMA***
```

Pela planilha eletrônica, obtém-se a recursão abaixo:

4	-17	4	8	-17

x1	x2	x3	x4	x5
0	0,235294118	0,249942779	0,249999999	0,25
5	4,173913043	4,007380925	4,000014471	4