DESENVOLVIMENTO DE UMA CALCULADORA FASORIAL

Gabriel Barbosa*

*12/0050935

Email: gabrielfbbarbosa@gmail.com

Resumo— 30 de Novembro de 2016 – Fasores trouxeram a solução para muitos problemas na área de eletromagnetismo. Entretanto, eles ainda exigem muitos cálculos quando temos que lidar com eles. Esse projeto foi desenvolvido com o objetivo de facilitar toda a matemática envolvida com fasores.

Palavras-chave— ferramenta, fasor, cálculo, visualização, operação

1 Introdução

O Professor Marco Terada, que ministra a aula de Eletromagnetismo 1, solicitou para a turma desenvolver apresentações e/ou projetos baseados na matéria ensinada e que pudessem de certa forma contribuir com a proposta sugerida. Enquanto alguns optaram por ir pelo lado científico e abordar fenômenos eletromagnéticos, peculiaridades, como podemos aproveitá-los e todo o raciocínio científico-matemático por trás deles, eu decidi ir por um caminho diferente. Eu optei pelo caminho de buscar algo prático que poderiam ajudar imediatamente estudantes e também porfissionais da área que necessitam de uma solução mais pragmática para cálculos que envolvem fasores.

Quando se está lidando com circuitos elétricos e existe uma incitação senoidal, a presença de fasores é indispensável. Aprender como eles funcionam, o que representam e como trabalhar com eles é de extrema importância. Entretanto, quando já se tem essa teoria por trás, infelizmente a dificuldade de trabalhar com fasores continua existindo. Eles vieram como um auxílio para resolver circuitos elétricos de excitação senoidal mas continuam sendo difíceis de lidar. Porquê então não facilitar o uso deles? E é aí que habita a minha proposta.

O que sugiro e inicio a implementação nesse projeto é uma ferramenta que, inicialmente, foi concebida para operações e visualizações de fasores. Com essa ferramenta, você poderá operar fasores com facilidade, visualizá-los em um gráfico Real x Imaginário e convertê-los entre suas várias possíveis visualizações. O estado atual do projeto é um protótipo semi-funcional com a estrutura praticamente pronta mas precisando de uma limpeza de código para remoção de erros e também por uma melhoria na interface gráfica para se utilizar todos os recursos existentes em seu código por trás.

Um dos requisitos mais importantes que levei em consideração na hora de tomar minhas decisões foi o fato de que desejo que isso seja uma ferramenta utilizada sem dificuldades e não importando que se esteja sendo usada. Esse requisito guiou desde as decisões de quais ferramentas eu usaria até sobre funcionalidades que implementaria. Essas decisões serão explicadas mais adiante com uma exposição não só das outras opções mas também o meu raciocínio por trás de cada decisão.

2 Os Fasores

O conceito de corrente alternada já existe a um bom tempo – o primeiro alternador data de 1832 – e teve vários avanços desde aquela época em diante, mas só foi no final daquele século que os fasores foram inventados e começaram a ser utilizados.

O nome fasor vem do inglês *phasor*, que é a junção de *phase* (fase) e *vector* (vetor), ou seja, um fasor significa também vetor de fase. Um fasor nada mais é do que uma representação usando números complexos de uma função senoidal mas com sua amplitude A, frequência angular ω e fase θ independentes uns dos outros e invariantes no tempo. A grande vantagem de usar fasores é que o uso deles transformam funções trigonométricas em funções algébricas, já que a frequência angular ω geralmente é um fator comum a todos os componentes num mesmo circuito.

2.1 Definição

O princípio básico no qual se baseia a existência dos fasores é uma representação vetorial bidimensional de um movimento harmônico simples. Esse tipo de movimento pode ser descrito pela equação

$$y(t) = A\cos(\omega t + \theta) \tag{1}$$

onde A é a amplitude, ω é a frequência angular e θ é a fase inicial. Se formos considerar uma representação no círculo geométrico do ponto $p(t)$, usando por base a equação 1, obteremos que

$$p(t) = A(\cos(\omega t + \theta)\overrightarrow{i} + \sin(\omega t + \theta)\overrightarrow{j}),$$

ou também

$$p(t) = (A\cos(\omega t + \theta), A\sin(\omega t + \theta)). \tag{2}$$

Vamos considerar também um vetor $\overrightarrow{v}(t)$ que simplesmente sai de $(0, 0)$ e aponta para $p(t)$. De

acordo com as formulações de Euler, funções senoidais podem ser representadas por uma combinação de funções exponenciais complexas. No nosso caso, o que mais nos interessa é a representação demonstrada na equação 3 abaixo que é criada com base na equação 2.

$$\overrightarrow{v}(t) = (\mathrm{Re}\{Ae^{i\theta}e^{i\omega t}\}, \mathrm{Im}\{Ae^{i\theta}e^{i\omega t}\}) \quad (3)$$

Como já foi abordado mais acima, o termo que envolve a frequência angular ω é o mesmo para todos os fasores do mesmo sistema sendo, portanto, ignorado. O que significa que o termo interno aos operadores Re e Im, $Ae^{i\theta}e^{i\omega t}$, pode ser escrito da forma simplificada $Ae^{i\theta}$.

2.2 Representações

Um fasor, assim como um vetor tradicionalmente falando, possui várias formas de ser representado. Cada forma favorece, ou enfatiza, uma característica diferente mas sempre representa a mesma coisa. Cada situação pode ser mais fácil de se lidar usando uma representação diferente, e é nesse ponto que a vantagem se torna desvantagem, pois, o fato de haver várias representações força que está trabalhando com fasores a constantemente converter de uma para outra e com rapidez. E converter de uma forma para a outra nem sempre é uma tarefa trivial. As formas de representação estão descritas a seguir.

Representação senoidal Essa é a representação usada e demonstrada na equação 1. Ela é a mais fácil de se entender quais são os efeitos práticos de cada operação matemática mas, em contrapartida, uma das mais trabalhosas de se fazer (dependendo da operação, obviamente).

$$\overrightarrow{v}(t) = A\cos(\omega t + \theta)$$

Representação exponencial Como discutido ao final da seção 2.1, essa representação vem das fórmulas de Euler aplicadas para outras formas de demonstrar uma fórmula senoidal usando fórmulas exponenciais. Elas são as que ficam no meio termo entre facilidade de uso e facilidade de compreensão.

$$\overrightarrow{v}(t) = Ae^{i\theta}e^{i\omega t}$$

ou

$$\overrightarrow{v} = Ae^{i\theta}$$

Representação angular Essa representação é a única das quatro que ainda não apareceu em nenhum lugar neste relatório. Ela é a mais compacta e simples de usar, contudo, para um olho desacostumado, ela pode induzir muito ao erro na hora de se operar com ela. Cuidado nunca é demais. Essa representação é a que representa maior facilidade de compreensão dentre as quatro, porém, como discutido, é a que pode causar maior confusão para quem não está acostumado.

$$\overrightarrow{v} = A\angle\theta$$

Representação complexa Essa forma também já apareceu anteriormente e ela aparenta ser mais complicada do que realmente é. A diferença é que ela usualmente só é usada quando o valor de t é dado. Ela pode ser usada sem essa condição mas geralmente é a mais díficil e é maior representação possível. Quando se tem um tempo t conhecido, ela pode se tornar bem pequenha. Ela é de extrema utilidade no caso desse relatório pois uma representação complexa pode ser facilmente transformada em um vetor bidimensional para plotagem em um gráfico.

$$\overrightarrow{v}(t) = A(\cos(\omega t + \theta) + i\sin(\omega t + \theta))$$

2.3 Aritmética

O principal propósito da criação dos fasores é permitir uma facilidade em lidar com circuitos de excitação senoidal. Portanto, eles não são usados só para visualização de características do circuito mas para cálculos. O que significa que, a existência de uma aritmética de fasor é imprescindível. Pelas características do fasor, suas operações podem ocorrer entre escalares complexos, entre fasores de mesma frequência e entre fasores de frequências diferentes.

Todas as operações demonstradas a seguir serão para fasores de frequências diferentes mas podem ser extrapoladas para fasores com frequências iguais. Simplificações para essas casos serão demonstradas, caso existam. Quando temos que operar com escalares complexos, basta considerá-los como fasores com frequência angular igual ao outro operando.

Adição Para obtermos $\overrightarrow{z}(t) = \overrightarrow{x}(t) + \overrightarrow{y}(t)$, a operação descrita abaixo precisa ocorrer.

$$\overrightarrow{z}(t) = A_x\cos(\omega_x t + \theta_x) + A_y\cos(\omega_y t + \theta_y)$$

Se $\omega_x = \omega_y$, podemos utilizarmos a visualização complexa para fazer a soma e depois retornar para a visualização senoidal, obtendo assim que

$$A_z = \sqrt{(A_x\cos\theta_x + A_y\cos\theta_y)^2 + (A_x\sin\theta_x + A_y\sin\theta_y)^2}$$

$$\theta_z = \arctan\left(\frac{A_x\sin\theta_x + A_y\sin\theta_y}{A_x\cos\theta_x + A_y\cos\theta_y}\right) \quad (4)$$

Subtração/Oposto A subtração não é nada além de uma soma com o oposto do segundo termo, $A - B = A + (-B)$, portanto, para se realizar uma subtração, basta calcularmos o oposto do segundo termo e realizar o mesmo procedimento

de uma soma. O oposto pode ser obtido de duas formas: invertendo o sinal da magnitude ou se somando 180º (ou π radianos) à fase. O que significa que se $\overrightarrow{z}(t) = -\overrightarrow{y}(t)$, obtemos que

$$\overrightarrow{z}(t) = -A_y \cos(\omega_y t + \theta_y) = A_y \cos(\omega_y t + \theta_y + \pi),$$

ou seja,

$$A_z = -A_y$$
$$\theta_z = \theta_y$$

ou

$$A_z = A_y$$
$$\theta_z = \theta_y + \pi \tag{5}$$

Multiplicação Se desejarmos calcular a multiplicação $\overrightarrow{z}(t) = \overrightarrow{x}(t) \times \overrightarrow{y}(t)$, o resultado é

$$\overrightarrow{z}(t) = A_x A_y \cos(\omega_x t + \theta_x) \cos(\omega_y t + \theta_y)$$

Se considerarmos $\omega_x = \omega_y$, a visualização exponencial será a melhor e mais fácil para se realizar a operação. Ao se transformar de volta para uma visualização senoidal, obtemos

$$A_z = A_x \times A_y$$
$$\theta_z = \theta_x + \theta_y \tag{6}$$

Divisão/Inversão A divisão, assim como a subtração, pode ser obtida de outras formas. No caso da divisão, é uma operação de multiplicação no qual o segundo elemento está invertido, ou seja, $\frac{A}{B} = A \times \frac{1}{B}$. Para se realizar a operação de inversão em um fasor, basta inverter a magnitude e inverter o sinal da inverter o sinal da fase, isto é, subtrair 360º (2π radianos) da fase atual. O que significa que se $\overrightarrow{z}(t) = \frac{1}{\overrightarrow{y}(t)}$, obtemos que

$$\overrightarrow{z}(t) = \frac{1}{A_y} \cos(\omega_y t + 2\pi - \theta_y),$$

ou seja,

$$A_z = \frac{1}{A_y}$$
$$\theta_z = 2\pi - \theta_y \tag{7}$$

Conjugação Complexa Essa operação é extremamente trivial de se fazer numa representação complexa e se traduz também de forma simples para as outras representações. Abaixo já está a representação senoidal da operação $\overrightarrow{z}(t) = \overrightarrow{y}^*(t)$.

$$\overrightarrow{z}(t) = A_y \cos(\omega_y t + 2\pi - \theta_y)$$

O que significa

$$A_z = A_y$$
$$\theta_z = 2\pi - \theta_y \tag{8}$$

Derivação A derivação é um procedimento mais complexo do que o abordado anteriormente mas pode ser realizado através da representação exponencial. Como $i\omega = \omega e^{i\frac{\pi}{2}}$, obtemos que, para $\overrightarrow{z}(t) = \frac{d(\overrightarrow{y}(t))}{dt}$,

$$\overrightarrow{z}(t) = \omega_y A_y \cos\left(\omega_y t + \theta_y + \frac{\pi}{2}\right)$$

Ou seja,

$$A_z = A_y \omega_y$$
$$\theta_z = \theta_y + \frac{\pi}{2} \tag{9}$$

Integração A integração pode ser vista, nesse contexto, como a operação inversa da derivada, da mesma forma que a divisão é a operação inversa da multiplicação e a subtração é a operação inversa da adição. Se formos realizar o mesmo procedimento que o tópico anterior mas para a integração, chegamos que basta multiplicar por $\frac{1}{i\omega} = \frac{1}{\omega e^{i\frac{\pi}{2}}}$. Isso significa que a operação $\overrightarrow{z}(t) = \int \overrightarrow{y}(t) dt$ resulta em

$$\overrightarrow{z}(t) = \frac{A_y}{\omega_y} \cos\left(\omega_y t + \theta_y - \frac{\pi}{2}\right)$$

Ou seja,

$$A_z = \frac{A_y}{\omega_y}$$
$$\theta_z = \theta_y - \frac{\pi}{2} \tag{10}$$

Quando se está lidando com fasores com frequências angulares ω diferentes, a aritmética pode ficar muito complexa e gerando sistemas com múltiplas oscilações senoidais simultâneas. Dependendo de quantos fasores de frequências diferentes forem somados, o resultado pode ser inviável, computacionalmente falando. Entretanto, ainda assim é possível é possível essas operações com facilidade por um computador, com uma restrição. A restrição é que basta o tempo t ser fornecido.

3 O Projeto

Inicialmente, também fazia parte da proposta do projeto poder calcular campos eletromagnéticos com pontas de prova em um ambiente 3D e usando cargas em objetos de tamanhos diversos. Essa proposta se tornou muito grande e, de fato, não possui muita ligação com o resto do projeto, que lida com fasores. É perceptível que, portanto, essas duas propostas podem ser divididas em dois projetos completamente diferentes com suas próprias dificuldades, teorias e aprendizados por trás. E esse é o motivo que me fez decidir por focar somente nos fasores.

Esse projeto está hospedado na íntegra na seguinte página no GitHub: `https://github.com/bestknighter/Calculadora-Fasorial`. Lá se encontra esse relatório, o código-fonte e alguns executáveis gerados para testes do protótipo.

3.1 Matemática

Para se realizar a implementação desse projeto, a parte matemática toda foi colocada no *namespace* MathCore. Esse *namespace* possui duas *structs*, Complexo e Fasor.

Complexo A *struct* Complexo possui funcionalidade única de dar suporte à representação retangular temporal e para facilitar a representação no plano Real x Imaginário na parte gráfica. A figura 1 mostra como é pequena a sua implementação, somente 48 linhas incluindo documentação.

```
public struct Complexo {

    /// Parte real...
    public readonly double Real;
    /// Parte imaginária...
    public readonly double Imaginario;
    /// Inicializa um número do tipo <see cref="Complexo"/>...
    public Complexo (double re, double im) {
        Real = re;
        Imaginario = im;
    }

    /// Retorna uma string que representa o objeto...
    public override string ToString () {
        string resp;
        if (Real != 0 && Imaginario != 0) {
            resp = (Imaginario>0) ? (Real + "+j" + Imaginario) : (Real + "-j" + (-1*Imaginario));

        } else if (Real != 0) {
            resp = "" + Real;

        } else if (Imaginario != 0) {
            resp = (Imaginario>0) ? ("j" + Imaginario) : ("-j" + (-1*Imaginario));

        } else {
            resp = "0";
        }
        return resp;
    }

}
```

Figura 1: Implementação da *struct* Complexo

Fasor A *struct* Fasor já é bem maior. Com 171 linhas incluindo documentação, essa struct é a responsável por toda e qualquer operação aritmética envolvendo fasores (figuras 3 e 4) e pela geração de strings para cada representação existente (figuras 6 e 5). O protótipo da struct pode ser encontrado na figura 2.

```
public struct Fasor {

    public readonly double Amplitude;
    public readonly double FrequenciaAngular;
    public readonly double FaseRadianos;

    /// Inicialize um <see cref="Fasor"/>...
    public Fasor (double amplitude, double frequenciaRadS, double faseRad) {
        if (amplitude < 0) {
            amplitude = -amplitude;
            faseRad += Math.PI;
        }
        faseRad %= 2 * Math.PI;

        Amplitude = amplitude;
        FrequenciaAngular = frequenciaRadS;
        FaseRadianos = (faseRad >= 0 ? faseRad : 2 * Math.PI + faseRad);
    }

    /// Representação complexa do fasor no tempo especificado...
    public Complexo Retangular (double tempo = 0) {
        return new Complexo (Amplitude*Math.Cos(FrequenciaAngular*tempo+FaseRadianos), Amplitude*Math.Sin(FrequenciaAngular*tempo+FaseRadianos));
    }
```

Figura 2: Protótipo de Fasor

Essa *struct* foi implementada com base na parte teórica deste relatório. Todas as funções e contas foram simplesmente uma implementação das equações da seção 2.3.

Por exemplo, a operação de adição (*operator + *na imagem 3) usa a equação 4. A operação de subtração (*operator - (lhs, rhs)* na imagem 3) usa a adição com o oposto (*operator - (lhs)* na imagem 3) do segundo termo da operação, 5. A operação de multiplicação (*operator * (lhs, rhs)* na imagem 3) usa a equação 6. A operação de divisão (*operator / (lhs, rhs)* na imagem 3) usa a multiplicação com o inverso (*Inverso* na imagem 3) do segundo termo da operação, 7.

```
public static Fasor operator + (Fasor lhs, Fasor rhs){
    double aux1 = lhs.Amplitude*Math.Cos (lhs.FaseRadianos) + rhs.Amplitude*Math.Cos (rhs.FaseRadianos);
    double aux2 = lhs.Amplitude*Math.Sin (lhs.FaseRadianos) + rhs.Amplitude*Math.Sin (rhs.FaseRadianos);

    double amp = Math.Sqrt (Math.Pow (aux1, 2) + Math.Pow (aux2, 2));
    double fase = Math.Atan2 (aux2, aux1);

    return new Fasor (amp, lhs.FrequenciaAngular, fase);
}

public static Fasor operator - (Fasor lhs) {
    return new Fasor (lhs.Amplitude, lhs.FrequenciaAngular, lhs.FaseRadianos + Math.PI);
}

public static Fasor operator - (Fasor lhs, Fasor rhs) {
    return lhs + (-rhs);
}

public static Fasor operator * (Fasor lhs, Fasor rhs) {
    return new Fasor (lhs.Amplitude * rhs.Amplitude, lhs.FrequenciaAngular, lhs.FaseRadianos + rhs.FaseRadianos);
}

/// Obtém o fasor inverso (1/fasor)...
public Fasor Inverso {
    get { return new Fasor (1 / Amplitude, FrequenciaAngular, 2 * Math.PI - FaseRadianos); }
}

public static Fasor operator / (Fasor lhs, Fasor rhs) {
    return lhs * rhs.Inverso;
}
```

Figura 3: Operações aritméticas

A operação conjugado complexo (*Conjugado* na imagem 4) usa a equação 8. A operação de derivação (*Derivado* na imagem 4) usa a equação 9. E a operação de integração (*Integrado* na imagem 4) usa a equação 10.

```
/// Obtém o fasor com a parte imaginária conjugada...
public Fasor Conjugado {
    get { return new Fasor (Amplitude, FrequenciaAngular, 2 * Math.PI - FaseRadianos); }
}

/// Obtém o fasor resultante da derivação...
public Fasor Derivado...

/// Obtém o fasor resultante da integração...
public Fasor Integrado {
    get { return new Fasor (Amplitude / FrequenciaAngular, FrequenciaAngular, FaseRadianos - (Math.PI / 2d)); }
}
```

Figura 4: Operações aritméticas

Com essas duas classes implementadas, a aplicação já pode ser construída da forma que se desejar e usando a ferramenta que quiser. Poderia ser OpenGL, WebGL, somente texto, Vulkan, Visual C#, ou qualquer outra API gráfica existente no mercado. Como desejo fazer o código uma vez e fazer a entrega de executáveis em várias plataformas ao mesmo tempo, sem me preocupar com otimizações, para esse protótipo, decidi utilizar a ferramenta Unity. A partir da próxima seção será abordado os específicos de como foi executada a implementação com essa ferramenta.

```
/// Representação retangular do fasor...
public string RepresentacaoRetangular (double tempo = 0) {
    return Retangular(tempo).ToString();
}

/// Representação exponencial do fasor...
public string RepresentacaoExponencial () {
    return Amplitude.ToString() + "e^i(" + FrequenciaAngular + "t+" + FaseRadianos + ")";
}

/// Representação angular do fasor...
public string RepresentacaoAngular() {
    return Amplitude.ToString() + "∠" + FaseRadianos*(180d/Math.PI) + "°";
}

/// Retorna uma string que representa o objeto...
public override string ToString () {
    return string.Format ("[Fasor: Amplitude={0}, Frequencia={1}, FaseRadianos={2}]", Amplitude, FrequenciaAngular, FaseRadianos);
}
```

Figura 5: Diversas representações por string

```
/// Representação cossenoide do fasor...
public string RepresentacaoCossenoide () {
    if (Amplitude == 0) {
        return "0";
    }

    string resp = Amplitude+"*cos(";

    //  Monta de forma bonita a string do termo do cosseno.
    if (FrequenciaAngular != 0 && FaseRadianos != 0) {
        if (FrequenciaAngular == 1) {
            resp += "t+" + FaseRadianos;
        } else {
            resp += FrequenciaAngular + "*t+" + FaseRadianos;
        }

    } else if (FrequenciaAngular == 1) {
        resp += "t";

    } else if (FrequenciaAngular != 0) {
        resp += FrequenciaAngular + "*t";

    } else if (FaseRadianos != 0) {
        resp += FaseRadianos;

    } else {
        resp += "0";
    }

    return resp+")";
}
```

Figura 6: Diversas representações por string

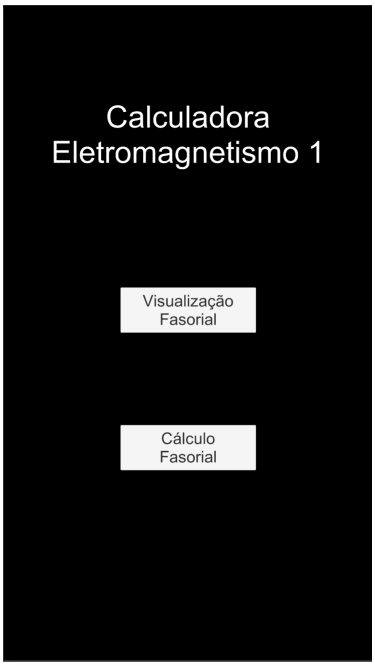

Figura 7: Menu Principal

3.2 Testes

Primeiramente, para garantir o bom funcionamento da parte mateática, foram implementados testes unitários. Esses testes puderam assegurar a integridade dos resultados durante sua implementação. Quanto a parte gráfica, não foram criados testes unitários pois eles mais dependem de testes, tentativas e erros, ao usar a interface.

3.3 Arquitetura

Como esse projeto foi proposto para ser feito em pouco tempo, somente um protótipo era planejado para o final. Portanto, a arquitura executada pode não ser a mais eficiente.

Nesse projeto, a interface gráfica é que controla tudo. É a interface gráfica que possui gerentes de fasores, que decide quando mudar de tela e quando fazer as contas.

Existem 3 cenas: Menu Principal, Visualização Fasorial e Cálculo Fasorial. Cada vez que se sai de uma cena, ao se retornar para ela, tudo será resetado. Da cena Menu Principal é possível ir para qualquer uma das outras cenas, mas cada cena só pode retornar para a cena principal.

Menu Principal Nessa cena, existem dois botões. Cada um te permite ir para cada uma das outras duas cenas. Ela existe para pré-carregar recursos necessários e permitir o usuário decidir o que ele deseja fazer.

Visualização Fasorial Em Visualização Fasorial, existe um menu-gaveta na parte de baixo da cena. Esse menu-gaveta é onde o usuário pode inserir ou remover fasores que serão mandados para a visualização no plano. Cada botão possui uma variável do tipo fasor que é onde é armazenado qual fasor aquele botão representa. É a função da classe responsável por desenhar no plano que pega esse fasor, obtêm o complexo no instante 0 e o plota no plano Real x Imaginário.

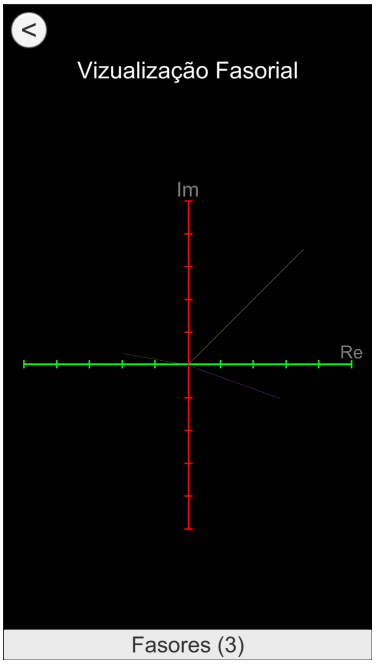

Figura 8: Visualização Fasorial

Cálculo Fasorial Essa cena é a calculadora propriamente dita. É aqui que o usuário pode executar a operação que desejar. Como esse projeto está em fase de protótipo, só é possível fazer con-

tas entre dois fasores e que possuem frequência angular iguais. Nesse caso, cada grupo de campos de inserção de dados é que possui um fasor e toda vez que um deles é terminado de ser editado, uma função é chamada que realiza a atualização do fasor resposta.

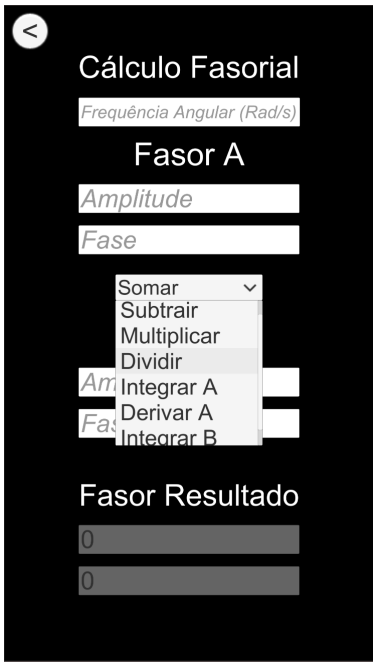

Figura 9: Calculadora Fasorial

3.4 Problemas e melhorias

Os únicos problemas que encontrei foram relacionados à ferramenta Unity e não à implementação da matemática por trás da calculadora. Até o presente o momento, os fasores só são visíveis na visualização quando se utiliza builds de desenvolvimento. Quando se utiliza versões de distrbuição as linhas simplesmente desaparecem. Usar entradas de toque ao invês do mouse+teclado possui comportamento instável, as vezes funcionando e as vezes não.

As melhorias podem ser diversas. Desde investimento em arte para o usuário ter uma boa experiência de uso até otimizações do aplicativo para redução de tamanho, redução de consumo de recursos e aumento de velocidade de execução. Dois destaques especiais são que a visualização fasorial poderia mudar com o tempo e ter linhas mais facilmente visíveis e que poderia ter uma opção para inserir um fasor e ter imediatamente sua conversão para todas as outras notações.

4 Conclusão

Ao final do projeto, foi possível perceber que a matemática por trás dos fasores não é tão complicada, mas ainda assim é trabalhosa de se fazer. Um produto finalizado e bem acabado que realiza o proposto por esse projeto com certeza trairá vantagens e vale a pena ser implementado. No caso disso ser feito, acredito que, devido à simplicidade do código para o cálculo, seja mais vantajoso usar ferramentas exclusivas de cada plataforma ao invés do Unity.

Após alguns testes usando a calculadora para resolver exercícios de circuitos elétricos com fontes senoidais, é perceptivel o ganho de tempo e agilidade para resolver as questões quando se está de posse da calculadora mesmo que seja um protótipo. É bem provável que um aplicativo bem acabado e com bom marketing seja popularmente usado por profissionais e estudantes que tenham que lidar com fasores constantemente.

Esse projeto não só tem muito onde melhorar mas também tem muitas funcionalidades que podem ser implementadas. E em ambos os casos, com relativa facilidade. O potencial de ser útil a muitos é inegavelmente visível e tangenciável.