Acadêmico: Ricardo Berri Mees

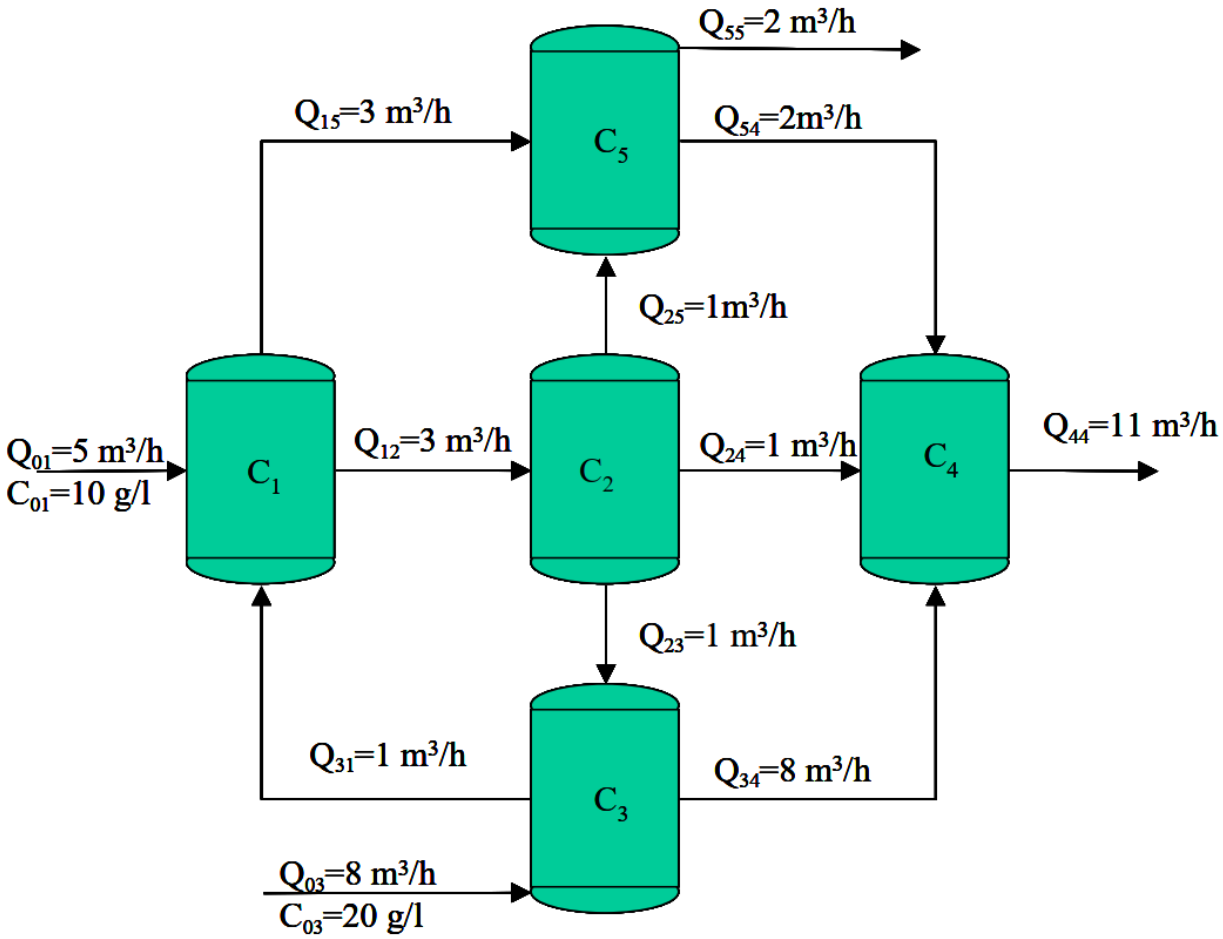
Disciplina: Modelos e Métodos Matemáticos em Engenharia Química

Sistemas de Equações Diferenciais Lineares e Não-Lineares

**Problema:**

Considere uma operação de mistura perfeita onde cinco tanques são conectados conforme esquema abaixo. Considere adiabáticos os tanques e as tubulações, e que o fluido tenha as mesmas propriedades da água com $T_{01}$= 70 C e $T_{03}$= 10 °C.

**Figura 1 –** Sistema de tanques.




Considere uma reação química de decomposição exotérmica da espécie genérica, cuja cinética pode ser expressa como $r = - kC^2$ onde $k = 10^3.exp((34{,}34-34222)/T)$ *L/*(g.min) e T em Kelvin. A energia liberada pela reação pode ser considerada constante e igual a $\Delta Hr = -80.000$ J / g. Os tanques possuem volumes de, respectivamente, 10, 5, 12, 6, e 15 m³.

Determine a evolução de $C_1$, $C_2$, $C_3$, $C_4$, $C_5$, $T_1$, $T_2$, $T_3$, $T_4$, $T_5$ em um período de 24h para um caso sem reação e um caso com reação. Considerar concentração inicial de 0 g/L e temperatura inicial de 20°C para todos os tanques. Utilize o método Runge-Kutta-Fehlberg para a solução.

**Solução:**

Deve-se, primeiro, desenvolver os balanços de massa e energia para cada tanque. Partindo da equação geral do balanço, temos que:

$$Acúmulo = Entrada - Saída + Geração$$

Para os balanços de massa e energia, respectivamente, temos que:

$$\frac{d(CiVi)}{dt} = \sum Qin * Cin - Ci * \sum Qout + rV \qquad \frac{d(TiVi)}{dt} = \sum Qin * Tin - Ti * \sum Qout + \frac{\Delta Hr.r.V}{\rho.Cp}$$

Como o volume é constante, pode-se simplificar o equacionamento para:

$$\frac{dCi}{dt} = \frac{\sum Qin * Cin - Ci * \sum Qout}{Vi} + r \qquad \frac{dTi}{dt} = \frac{\sum Qin * Tin - Ti * \sum Qout}{Vi} + \frac{\Delta Hr.r}{\rho.Cp}$$

Conforme fornecido pelo problema, a geração será dada por $r = -kC^2$, portanto, temos que:

$$\frac{dCi}{dt} = \frac{\sum Qin * Cin - Ci * \sum Qout}{Vi} - k.Ci^2 \qquad \frac{dTi}{dt} = \frac{\sum Qin * Tin - Ti * \sum Qout}{Vi} - \frac{\Delta Hr.k.Ci^2}{\rho.Cp}$$

Onde *k* é igual a:

$$k = \frac{50}{3} \times e^{\left(-34187{,}66/_T\right)} \qquad k = 0$$

** Para o caso com reação* ** Para o caso sem reação*

Onde:

Q=Vazão Volumétrica [m³/s];

C=Concentração [kg/m³];

T=Temperatura [K];

k=Constante de Velocidade de Reação [m³/kg.s];

ΔHr=Energia liberada pela Reação [J/kg];

ρ=Massa Específica do Fluido [kg/m³];

Cp=Calor Específico do Fluido [J/kg.K];

In = Entrada;

Out= Saída;

i = Índice do tanque.

Para resolver o sistema de equações diferenciais, foi escrito um código na linguagem de programação Javscript para a solução do problema utilizando o método de Runge-Kutta-Fehlberg (RKF).

O método RKF é um esquema numérico utilizado para resolver equações diferenciais ordinárias (EDOs). Ele é uma variante do método clássico de Runge-Kutta, mas com a adição de estimativas de erro embutidas e um tamanho de passo adaptativo. Esse método opera em um intervalo específico e avança a solução da EDO a partir de um ponto inicial para o próximo ponto dentro desse intervalo. Para isso, utiliza uma combinação ponderada de gradientes da função derivada em vários pontos intermediários, aproximando a mudança na solução ao longo do intervalo.

As equações de Ks são essenciais no método RKF, pois são utilizadas para calcular as inclinações nos pontos intermediários, auxiliando na aproximação da mudança na solução da EDO. Essas equações são derivadas dos coeficientes do método e fornecem os valores de K1, K2, K3, K4, K5, e K6. Cada um desses valores é utilizado para avançar a solução da EDO em cada etapa. O número de equações de Ks é definido de acordo com o número de etapas do método RKF utilizado. No caso específico do RKF de quinta ordem, que será aplicado neste trabalho, temos seis equações de Ks:

$$k1 = h \times f(t + 0h, y)$$

$$k2 = h \times f\left(t + \frac{1}{4}h, y + \frac{1}{4}k1\right)$$

$$k3 = h \times f\left(t + \frac{3}{8}h, y + \frac{3}{32}k1 + \frac{9}{32}k2\right)$$

$$k4 = h \times f\left(t + \frac{12}{13}h, y + \frac{1932}{2197}k1 - \frac{7200}{2197}k2 + \frac{7296}{2197}k3\right)$$

$$k5 = h \times f\left(t + 1h, y + \frac{439}{216}k1 - 8k2 + \frac{3680}{513}k3 - \frac{845}{4104}k4\right)$$

$$k6 = h \times f\left(t + \frac{1}{2}h, y - \frac{8}{27}k1 + 2k2 - \frac{3544}{2565}k3 + \frac{1859}{4104}k4 - \frac{11}{40}k5\right)$$

Nessas equações, f representa a função derivada que define a EDO, t é a variável independente, y é o vetor de variáveis dependentes e h é o tamanho do passo. No método RKF, além das equações de Ks mencionadas anteriormente, o esquema numérico utiliza uma combinação ponderada das inclinações calculadas para obter duas aproximações diferentes da solução da EDO. Essas aproximações são conhecidas como aproximação de quarta ordem (y4) e aproximação de quinta ordem (y5).

As aproximações de quarta ordem (y4) e quinta ordem (y5) são calculadas da seguinte forma:

$$y4 = y + \frac{25}{216}k1 + \frac{1408}{2565}k3 + \frac{2197}{4104}k4 - \frac{1}{5}k5$$

$$y5 = y + \frac{16}{135}k1 + \frac{6656}{12825}k3 + \frac{28561}{56430}k4 - \frac{9}{50}k5 + \frac{2}{55}k6$$

Nessa equações, y representa a solução da EDO no ponto atual, e K1, K3, K4, K5, e K6 são as inclinações calculadas nas equações de Ks. Na aproximação de quinta ordem, além das inclinações K1, K3, K4 e K5, também é levada em consideração a inclinação K6, obtida na sexta equação de Ks. Portanto, a aproximação de quinta ordem se torna mais precisa que a de quarta ordem.

Ao comparar as duas aproximações, o método RKF determina o erro estimado para decidir se deve aumentar ou diminuir o tamanho do passo na próxima iteração. Isso permite que o método se adapte às características da EDO e forneça uma solução numérica precisa e eficiente ao longo do intervalo desejado. Este erro entre soluções pode ser determinado com a seguinte equação:

$$E = \frac{1}{360}k1 - \frac{128}{4275}k3 - \frac{2197}{75240}k4 + \frac{1}{50}k5 + \frac{2}{55}k6$$

Em resumo, o método Runge-Kutta-Fehlberg (RKF) é uma abordagem numérica eficaz para resolver equações diferenciais ordinárias. Suas equações de Ks e as aproximações de quarta e quinta ordem garantem uma solução precisa e adaptativa. Agora, com o entendimento teórico do método, prossegue-se para a próxima etapa: a implementação do algoritmo RKF em JavaScript.

Na implementação, os coeficientes foram estruturados em *arrays* para facilitar a manipulação das equações em um *loop*. O *array* A contém os coeficientes de ponderação que são multiplicados pelo tamanho do passo e adicionados ao tempo para avaliar as funções derivadas. O *array* B, uma matriz bidimensional, contém os coeficientes que ponderam os valores de K, que são os gradientes da função derivada calculados em pontos intermediários. Esses coeficientes são então somados às variáveis dependentes. Os *arrays* C e CH são utilizados para calcular as soluções de 4ª e 5ª ordem, respectivamente. Eles consistem em coeficientes ponderados que multiplicam os valores de K correspondentes e são somados às variáveis dependentes para obter as aproximações da solução. Por fim, o *array* CT é responsável pelos coeficientes utilizados no cálculo do erro entre as soluções de 4ª e 5ª ordem. Esses coeficientes ponderam as diferenças entre as estimativas de 4ª e 5ª ordem, permitindo a determinação do erro e o ajuste adequado do tamanho do passo.

```
const   A = [0, 1 / 4, 3 / 8, 12 / 13, 1, 1 / 2],
        B = [   [0, 0, 0, 0, 0],
                [1 / 4, 0, 0, 0, 0],
                [3 / 32, 9 / 32, 0, 0, 0],
                [1932 / 2197, -7200 / 2197, 7296 / 2197, 0, 0],
                [439 / 216, -8, 3680 / 513, -845 / 4104, 0],
                [-8 / 27, 2, -3544 / 2565, 1859 / 4104, -11 / 40]
    ],  C = [25 / 216, 0, 1408 / 2565, 2197 / 4104, -1 / 5, 0],
        CH= [16 / 135, 0, 6656 / 12825, 28561 / 56430, -9 / 50, 2 / 55]
        CT= [-1 / 360, 0, 128 / 4275, 2197 / 75240, -1 / 50, -2 / 55];
```

Seguidamente, as variáveis da iteração foram iniciadas, tomando para si os valores iniciais.

```
let t = t0, y = [...y0], h = h0;
```

Depois disso, foi iniciado o *loop* com a condição de execução enquanto o tempo atual for menor que o tempo final desejado.

```
while(t < tf) {
      // Código //

}
```

Dentro do loop, foi definido um limite superior para o passo, sendo o menor valor entre o próprio passo, a diferença entre o tempo atual e o tempo final, e o tempo de interesse simulado dividido por 1000. Também inicializou-se os *arrays* de Ks (com dimensões 6 x N° funções) e Erro (com dimensão de N° de funções).

```
        h = Math.min(h, tf - t, 0.001*(tf - t0))

        let k = Array.from({ length: 6 }, () => new Array(f.length)),
            TE = new Array(f.length).fill(0);
```

Em seguida, foram calculados os valores dos Ks e dos Erros para cada função.

```
    for(let i=0;i<6;i++){
        for(let j=0;j<f.length;j++){
            k[i][j] = h * f[j](t+A[i]*h, y.map((yi) => {return B[i].reduce((sum, Bm, m) =>
                  { return sum + Bm * (k[m] ? (k[m][j] || 0) : 0)}, yi)}))
            TE[j] += CT[i] * k[i][j];
        }
    }
```

Por fim, o maior erro é comparado com a tolerância definida. Em caso afirmativo, o tempo é atualizado, as variáveis dependentes são substituídas pelo valor de 5° ordem, e o passo é multiplicado por 3. Em caso negativo, o passo é dividido pela metade.

```
if(Math.max(...TE.map(Math.abs)) < tol) { t += h;
    for(let j=0; j<f.length; j++){
        y[j] = CH.reduce( (sum, CHm, m)=> {return (sum + CHm*k[m][j])}, y[j])
    } // Resultado de 5º Ordem
    h *= 3;
} else { h *= 0.5;}
```

Feito a função RKF para solução do sistema de equações, seguiu-se com a criação de classes para estruturar as informações do problema. Foi idealizado 4 classes, sendo duas com informações estáticas e duas com informações de instância.

As classes estáticas estão apresentadas abaixo:

```
class Water {
    static ρ = 994; // kg/m³
    static Cp=4186; // J/kg.°C
}

class Reaction {
    static ΔHr() {return -80000000} // J/kg
    static k(Temperature) {return useReaction ?
                                (Math.exp(-34187.66 / (Temperature)) * 1000 / 60) :
                                0
      } // m³/kg.s
}
```

As outras duas classes, de instância, são as classes *Tanque* e *Pipe*.

A classe *Pipe* possui uma função construtora que define a origem e o destino de uma tubulação, e a vazão volumétrica passando por ela. A vazão volumétrica deve ser fornecida em m³/h, para que seja convertida em m³/s.

```
class Pipe {
    constructor(origem, destino, flow) {
        this.i = origem; // inicial
        this.f = destino; // final
        this.Q = (flow/3600); // m³/s
    }
}
```

A classe *Tanque* possui uma função construtora que define o nome e as variáveis do Tanque, concentração, temperatura, e volume. Caso não seja fornecido os valores, tratra-se como valor padrão de concentração, temperatura e volume, os valores 0 kg/m³, 20 °C, e 1 m³ respectivamente. O valor de temperatura é transformado para Kelvin para que possa ser utilizado nas equações sem nenhuma transformação.

```
class Tanque {
    constructor(name, concentration = 0, temperature = 20, volume = 1) {
        this.name = name;
        this.C = concentration; // kg/m³
        this.T = (temperature+273.15); // K
        this.V = volume; // m³
    }
}
```

Além da função construtora, a classe tanque também possui outras duas funções para cálculo das equações diferenciais de concentração e temperatura. Para obter os valores de somatória de entrada e saída de concentração e temperatura do tanque, todas as tubulações definidas no programa são iteradas, buscando aquelas com valor de origem e destino no tanque do qual a função foi chamada. Para que estas funções possam ser utilizadas pelo RKF, elas tomam como parâmetros de entrada o tempo e um *array* de variáveis y.

```
calcularDerivadaConcentracao(t, y){
    let somaEntradaC = 0, somaSaidaC = 0;
    let Ti = y[this.Tindex], Ci = y[this.Cindex], k = Reaction.k(Ti);
    for (let i = 0; i < pipes.length; i++) {
        const pipe = pipes[i];
        somaEntradaC += pipe.f === this.name ? pipe.Q * y[tanque[pipe.i].Cindex] : 0;
        somaSaidaC += pipe.i === this.name ? pipe.Q * Ci : 0;
    }
    let dCdt = (((somaEntradaC - somaSaidaC) / this.V) - (k * Ci * Ci));
    return dCdt;
}

calcularDerivadaTemperatura(t, y){
    let somaEntradaT = 0, somaSaidaT = 0;
    let Ti = y[this.Tindex], Ci = y[this.Cindex], k = Reaction.k(Ti);
    for (let i = 0; i < pipes.length; i++) {
        const pipe = pipes[i];
        somaEntradaT += pipe.f === this.name ? pipe.Q * y[tanque[pipe.i].Tindex] : 0;
        somaSaidaT += pipe.i === this.name ? pipe.Q * Ti : 0;
    }
    let dTdt = ((somaEntradaT - somaSaidaT)/(this.V)
                - (Reaction.ΔHr() * k * Ci* Ci)/(Water.ρ * Water.Cp))
    return dTdt;
}
```

A entrada de dados foi feita através da criação de um objeto de Tanques e um *array* de Pipes, fornecendo os dados conforme o caso a ser resolvido. Segue abaixo código da inicialização destas variáveis:

```
// Entrada de Dados
const tanque = {
    inf1: new Tanque("inf1", 10, 70), inf3: new Tanque("inf3", 20, 10),
    1: new Tanque(1, undefined, undefined, 10),
    2: new Tanque(2, undefined, undefined, 5),
    3: new Tanque(3, undefined, undefined, 12),
    4: new Tanque(4, undefined, undefined, 6),
    5: new Tanque(5, undefined, undefined, 15),
    inf4: new Tanque("inf4"), inf5: new Tanque("inf5")
}

const pipes = [
    new Pipe("inf1", 1, 5), new Pipe("inf3", 3, 8), new Pipe(1, 2, 3),
    new Pipe(1, 5, 3), new Pipe(2, 3, 1), new Pipe(2, 4, 1),
    new Pipe(2, 5, 1), new Pipe(3, 1, 1), new Pipe(3, 4, 8),
    new Pipe(4, "inf4", 11), new Pipe(5, 4, 2), new Pipe(5, "inf5", 2),
]
```

Após executar o código do programa, obteve-se os resultados abaixo, para cada caso.

**Caso 01**: Sem reação

**Figura 2 –** Evolução temporal da concentração e temperatura para o caso 01.

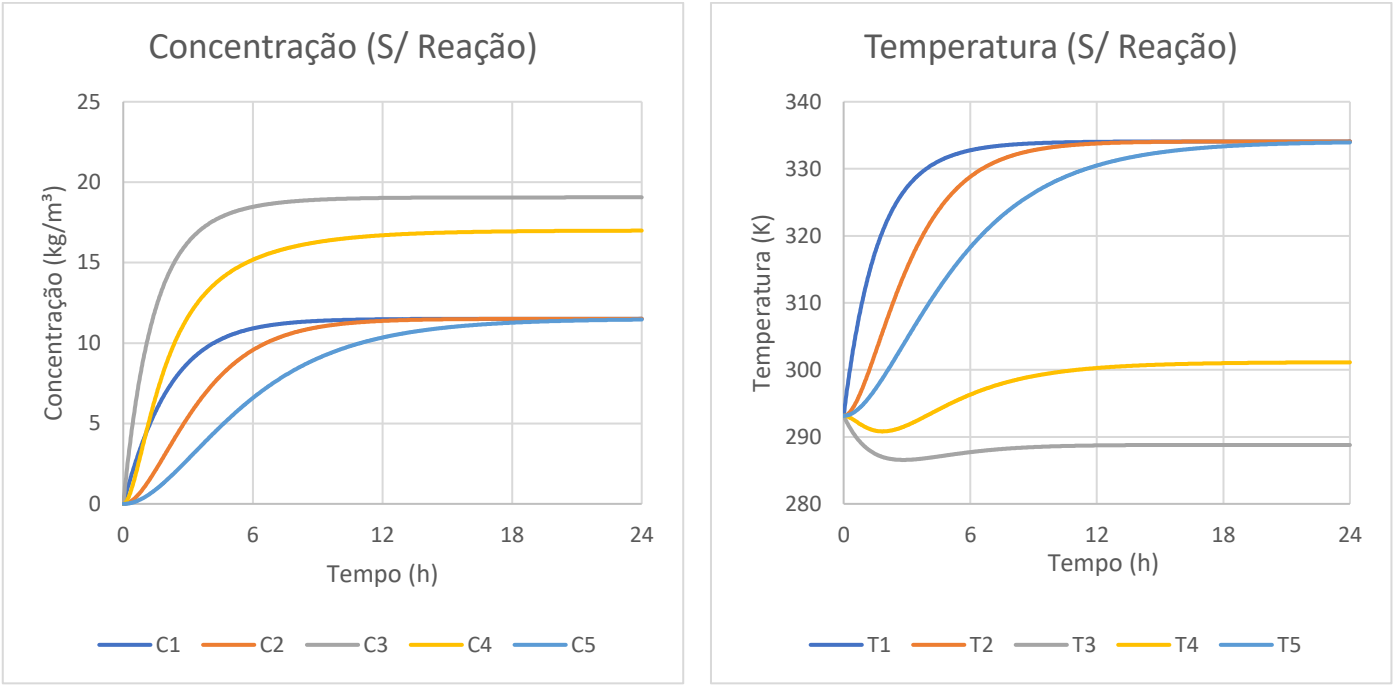



**Tabela 1 –** Condições finais para o caso 01.

| Tanque | Concentração [kg/m³] | Temperatura [°C] |
|---|---|---|
| 1 | 11.509327 | 60.943147 |
| 2 | 11.508950 | 60.942269 |
| 3 | 19.056462 | 15.660045 |
| 4 | 16.987486 | 27.977508 |
| 5 | 11.459569 | 60.791395 |

**Caso 02**: Com reação

**Figura 3 –** Evolução temporal da concentração e temperatura para o caso 01.

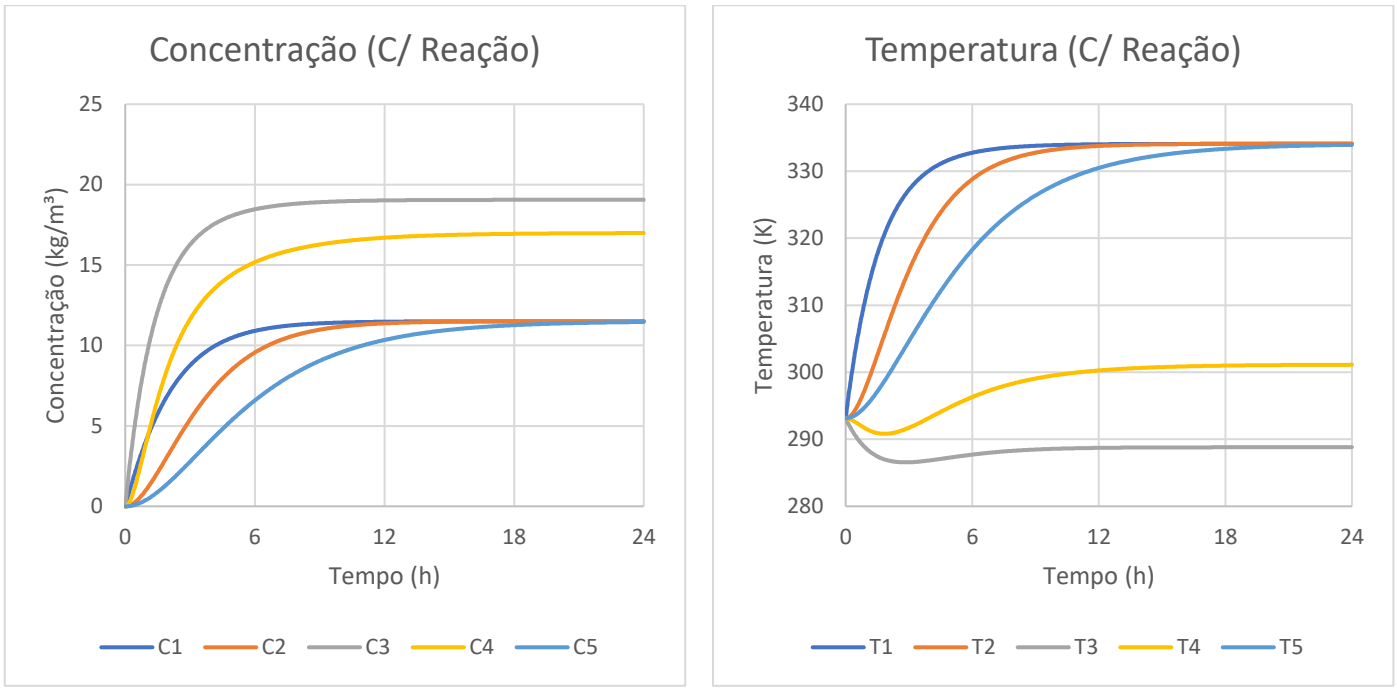


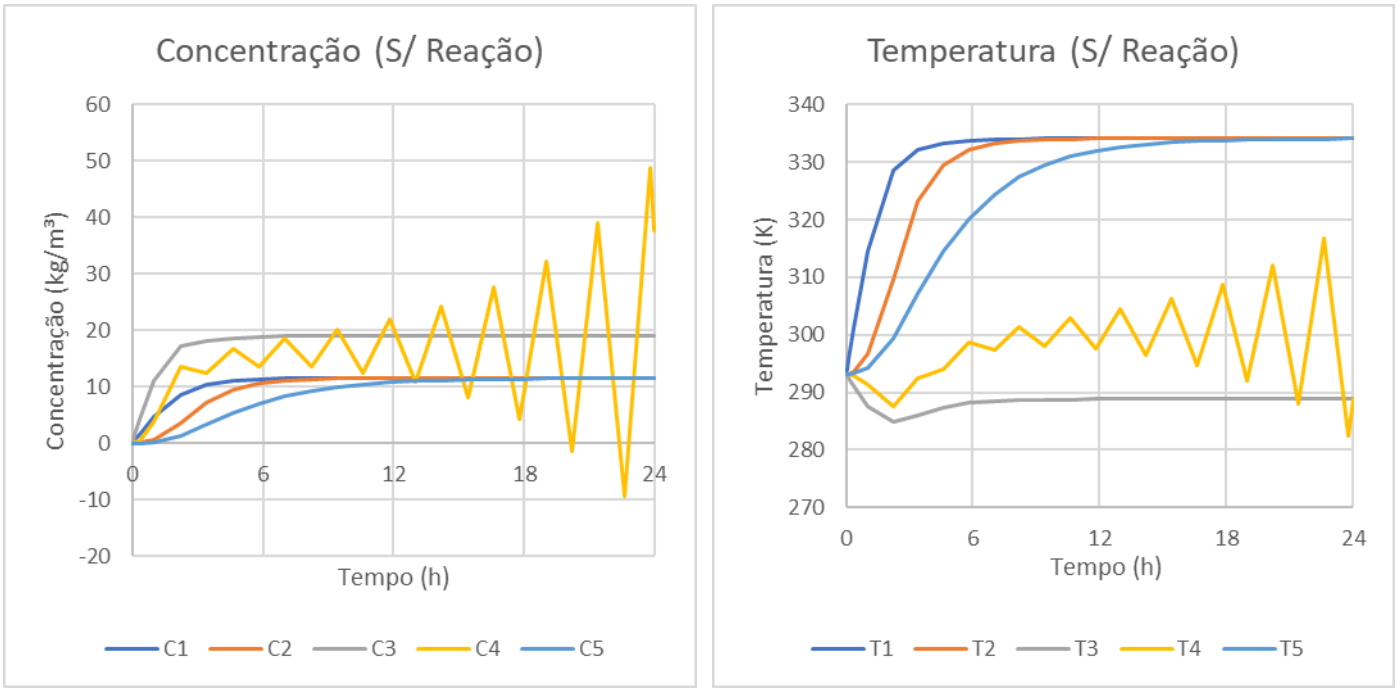

**Tabela 2 –** Condições finais para o caso 02.

| Tanque | Concentração [kg/m³] | Temperatura [°C] |
|---|---|---|
| 1 | 11.509327 | 60.943147 |
| 2 | 11.508950 | 60.942269 |
| 3 | 19.056462 | 15.660045 |
| 4 | 16.987486 | 27.977508 |
| 5 | 11.459569 | 60.791395 |

**Considerações Finais**:

Foi obtido êxito na implementação do método Runge-Kutta-Fehlberg para a solução de sistemas de equações diferenciais lineares (Sem reação) e não lineares (Com reação). Além do aprofundamento na parte teórica do método, também foi possível analisar o impacto dos parâmetros de tolerância e passo, conforme pode-se notar na figura 4, onde um limite superior de passo muito grande gerou uma solução com alta divergência do valor real para um dos tanques, embora a tolerância entre estimativas de soluções estava sendo atendida.

**Figura 4 –** Divergência de solução devido a falha de ajuste fino de parâmetros do RKF.




Quanto aos resultados, nota-se a proximidade de respostas em ambos os casos. Embora o segundo caso esteja considerando uma reação, o valor obtido no exponencial da constante de velocidade de reação é muito baixo, fazendo com que o termo de reação frente à entrada e saída de fluidos seja irrisório. Foi implementado, no código, um log para verificar os valores das taxas de entrada e saída de fluido, e de reação, e pode-se perceber a diferença na grandeza dos valores para todos os tanques:

| | | |
|---|---|---|
| dC1/dt= | I/O: 0.0012517867090131014 | r: 1.1294913927113991e-48 |
| dC2/dt= | I/O: 0.00021781602503223645 | r: 6.5045268035113565e-52 |
| dC3/dt= | I/O: 0.0029594129771546504 | r: 2.502335277418227e-49 |
| dC4/dt= | I/O: 0.0010370981848899074 | r: 1.2613055655248752e-50 |
| dC5/dt= | I/O: 0.00007807162238265699 | r: 6.850417516954046e-53 |

Abaixo segue código completo, disponível para download em github.com/Dekayra/AulaModelagem

Arquivo “rkf.js”

```
function rkf(f, y0, t0, h0, tf, resultados = [], logCallback) {

    const   A = [0, 1 / 4, 3 / 8, 12 / 13, 1, 1 / 2],
            B = [   [0, 0, 0, 0, 0],
                    [1 / 4, 0, 0, 0, 0],
                    [3 / 32, 9 / 32, 0, 0, 0],
                    [1932 / 2197, -7200 / 2197, 7296 / 2197, 0, 0],
                    [439 / 216, -8, 3680 / 513, -845 / 4104, 0],
                    [-8 / 27, 2, -3544 / 2565, 1859 / 4104, -11 / 40]
        ],  C = [25 / 216, 0, 1408 / 2565, 2197 / 4104, -1 / 5, 0],
            CH= [16 / 135, 0, 6656 / 12825, 28561 / 56430, -9 / 50, 2 / 55]
            CT= [-1 / 360, 0, 128 / 4275, 2197 / 75240, -1 / 50, -2 / 55];

    let t = t0, y = [...y0], h = h0,
    itSuc=0, itTot=0, itMax = 50000000;

    while(t < tf && (itTot < itMax || 100*t/tf > 90)) {
        h = Math.min(h, tf - t, 0.001*(tf - t0))

        let k = Array.from({ length: 6 }, () => new Array(f.length)),
            TE = new Array(f.length).fill(0);

        for(let i=0;i<6;i++){
            for(let j=0;j<f.length;j++){
                k[i][j] = h * f[j](t+A[i]*h, y.map((yi) => {return B[i].reduce((sum, Bm,
m) => { return sum + Bm * (k[m] ? (k[m][j] || 0) : 0)}, yi)}))
                TE[j] += CT[i] * k[i][j];
            }
        }

        // let y_4 = new Array(f.length), y_5 = new Array(f.length);
        // for(let j=0; j<f.length; j++){
        //     y_4[j] =  C.reduce( (sum, CHm, m)=> {return (sum + CHm*k[m][j])}, y[j])
        //     y_5[j] = CH.reduce( (sum, CHm, m)=> {return (sum + CHm*k[m][j])}, y[j])
        // }

        if(Math.max(...TE.map(Math.abs)) < tol) { t += h;
            for(let j=0; j<f.length; j++){
                y[j] = CH.reduce( (sum, CHm, m)=> {return (sum + CHm*k[m][j])}, y[j])
            } // Resultado de 5° Ordem

            logCallback({ t, y: [...y] }, {t, h, itSuc, itTot, itMax})

                    h *= 3;    itSuc++;
        } else {    h *= 0.5;} itTot++;
    } return resultados;
}

module.exports = rkf;
```

Arquivo “codigo.js”

```
const rkf = require('./rkf.js');

const fs = require('fs'); function salvarResultados(resultados) {
    const headers = ['Tempo'];
    for (let i = 1; i <= resultados[0].y.length / 2; i++) {
      headers.push(`C${i}`, `T${i}`);
    }

    const data = resultados.map((resultado) => {
      const t = resultado.t ? (resultado.t/3600).toString().replace('.', ',') : 0;
      const y = resultado.y.map((valor) => valor.toString().replace('.', ','));
      return [t, ...y];
    });

    const csvData = [headers, ...data];
    const csvContent = csvData.map((row) => row.join(';')).join('\n');
    fs.writeFileSync(`resultados_${useReaction ? "cr" : "sr"}.csv`, csvContent);
}

class Tanque {
    constructor(name, concentration = 0, temperature = 20, volume = 1) {
        this.name = name;
        this.C = concentration; // kg/m³
        this.T = (temperature+273.15); // K
        this.V = volume; // m³
    }

    calcularDerivadaConcentracao(t, y){
        let somaEntradaC = 0, somaSaidaC = 0;
        let Ti = y[this.Tindex], Ci = y[this.Cindex], k = Reaction.k(Ti);
        for (let i = 0; i < pipes.length; i++) {
            const pipe = pipes[i];
            somaEntradaC += pipe.f === this.name ? pipe.Q * y[tanque[pipe.i].Cindex] : 0;
            somaSaidaC += pipe.i === this.name ? pipe.Q * Ci : 0;
        }
        let dCdt = (((somaEntradaC - somaSaidaC) / this.V) - (k * Ci * Ci));
        return dCdt;
    }

    calcularDerivadaTemperatura(t, y){
        let somaEntradaT = 0, somaSaidaT = 0;
        let Ti = y[this.Tindex], Ci = y[this.Cindex], k = Reaction.k(Ti);
        for (let i = 0; i < pipes.length; i++) {
            const pipe = pipes[i];
            somaEntradaT += pipe.f === this.name ? pipe.Q * y[tanque[pipe.i].Tindex] : 0;
            somaSaidaT += pipe.i === this.name ? pipe.Q * Ti : 0;
        }
        let dTdt = ((somaEntradaT - somaSaidaT)/(this.V) - (Reaction.ΔHr() * k * Ci*
Ci)/(Water.ρ * Water.Cp))
        return dTdt;
    }
}
```

```
class Pipe {
    constructor(origem, destino, flow) {
        this.i = origem; // inicial
        this.f = destino; // final
        this.Q = (flow/3600); // m³/s
    }
}

class Water {
    static ρ = 994; // kg/m³
    static Cp=4186; // J/kg.°C
}

class Reaction {
    static R = 8.314 // J/mol.K
    static ΔHr() {return -80000000} // J/kg
    static k(Temperature) {return useReaction ? (Math.exp(-34187.66 / (Temperature)) *
1000 / 60) : 0} // m³/kg.s
}

function realizarCalculos(){
    let f = [], y0 = [], resultados = [];
    let logTimeLast=0, logTimeInterval=10, logTime = 0;
    let tempoInicio = Date.now();

    console.log(`[i] Preparando condições iniciais.`); // Escrever as funções e valores
iniciais
    for (let key in tanque) {
        if (!isNaN(parseFloat(key)) && isFinite(key)) {
          y0.push(tanque[key].C); tanque[key].Cindex = y0.length - 1
          f.push((t, y) => tanque[key].calcularDerivadaConcentracao(t, y))
          y0.push(tanque[key].T); tanque[key].Tindex = y0.length - 1
          f.push((t, y) => tanque[key].calcularDerivadaTemperatura(t, y))
        }
    }
    for (let key in tanque) {
        if (isNaN(key)) {
            y0.push(tanque[key].C); tanque[key].Cindex = y0.length - 1
            y0.push(tanque[key].T); tanque[key].Tindex = y0.length - 1
        }
    }
    resultados.push({ t: t0, y: [...y0] }); // Salvar as condições iniciais

    console.log(`[i] Iniciando RKF.`); // Iniciar RKF
    rkf(f, y0, t0, h0, tf, resultados, logCallback)

    logCallback(resultados[resultados.length-1], undefined, false)

    console.log(`[i] Salvando Resultados.`); // Escrever resultados em um arquivo
    salvarResultados(resultados)

    console.log(`[i] Finalizado. Tempo de execução: ${(Date.now() - tempoInicio)/1000}s`);
```

```
    return resultados

    function logCallback(resultado, info = {}, isRKF = true) {
        if(info.t - logTimeLast > logTimeInterval || !isRKF){ logTimeLast = info.t
            resultados.push(resultado);
        }
        if(Date.now() - logTime > 250 || !isRKF) {logTime = Date.now()
            if(isRKF){
                console.log(`i= ${info.itSuc}/${info.itTot}/${info.itMax}`);
                console.log(`t=
${(info.t).toFixed(4)}s\t\t${(info.t*100/tf).toFixed(4)}%\t\th=${info.h.toFixed(4)}`);
            } else { console.log(`Condições Finais:`)}
            console.log(`Tanque 1: C=${resultado.y[0].toFixed(6)} T=${(resultado.y[1]-
273.15).toFixed(6)} °C`)
            console.log(`Tanque 2: C=${resultado.y[2].toFixed(6)} T=${(resultado.y[3]-
273.15).toFixed(6)} °C`)
            console.log(`Tanque 3: C=${resultado.y[4].toFixed(6)} T=${(resultado.y[5]-
273.15).toFixed(6)} °C`)
            console.log(`Tanque 4: C=${resultado.y[6].toFixed(6)} T=${(resultado.y[7]-
273.15).toFixed(6)} °C`)
            console.log(`Tanque 5: C=${resultado.y[8].toFixed(6)} T=${(resultado.y[9]-
273.15).toFixed(6)} °C`)
            console.log('------------------------');
        }
    }
}

// Entrada de Dados
const tanque = {
    inf1: new Tanque("inf1", 10, 70),
    inf3: new Tanque("inf3", 20, 10),
    1: new Tanque(1, undefined, undefined, 10),
    2: new Tanque(2, undefined, undefined, 5),
    3: new Tanque(3, undefined, undefined, 12),
    4: new Tanque(4, undefined, undefined, 6),
    5: new Tanque(5, undefined, undefined, 15),
    inf4: new Tanque("inf4"),
    inf5: new Tanque("inf5")
}

const pipes = [
    new Pipe("inf1", 1, 5), new Pipe("inf3", 3, 8), new Pipe(1, 2, 3),
    new Pipe(1, 5, 3), new Pipe(2, 3, 1), new Pipe(2, 4, 1),
    new Pipe(2, 5, 1), new Pipe(3, 1, 1), new Pipe(3, 4, 8),
    new Pipe(4, "inf4", 11), new Pipe(5, 4, 2), new Pipe(5, "inf5", 2),
]

global.t0 = 0;
global.tf = 24*3600;
global.h0 = 1;
global.tol= 0.0000000001;
global.useReaction = true;

realizarCalculos();
```