## Метод Ньютона для системы нелинейных уравнений с постоянной матрицей Якоби

Метод Ньютона используется для итеративного решения систем нелинейных уравнений вида

$F(x) = 0$, где $F(x) = [f_1(x), f_2(x), \ldots, f_n(x)]$,

на каждом шаге нужно подставлять текущее приближение переменных x, в каждое уравнение системы и получать их значения.

$$F(x) = \begin{bmatrix} f_1(x) \\ f_2(x) \\ \vdots \\ f_n(x) \end{bmatrix}$$

```
function evaluateSystem(equations, variables) {
    return equations.map(fn => fn(...variables));
}
```

Здесь `equations` — массив функций, которые представляют уравнения, а `variables` — текущее значение переменных $x$.

Функция возвращает массив значений функций при подстановке текущих переменных.

### **Формула 2: Численное вычисление якобиана $J(x)J(x)$J(x)**

Якобиан $J(x)J(x)$J(x) — это матрица частных производных функций системы по каждой переменной:

$$J(x) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}$$

Для численного вычисления производных используем метод конечных разностей:

$$\frac{\partial f_i}{\partial x_j} \approx \frac{f_i(x_1, x_2, \ldots, x_j + \delta, \ldots, x_n) - f_i(x)}{\delta}$$

где $\delta$ — небольшое значение (в коде это `delta = 1e-5`)

```
function computeJacobian(equations, variables, delta = 1e-5) {
    const numVars = variables.length;
    const jacobian = Array.from({ length: equations.length }, () => new Array(numVars).fil

    for (let i = 0; i < equations.length; i++) {
        for (let j = 0; j < numVars; j++) {
            const variablesDelta = [...variables];
            variablesDelta[j] += delta;
            const f1 = equations[i](...variablesDelta);
            const f0 = equations[i](...variables);
            jacobian[i][j] = (f1 - f0) / delta;
        }
    }
    return jacobian;
}
```

### Формула 3: Итеративное обновление переменных

После вычисления текущих значений функций *F(x)F(x)*F(x) и матрицы якобиана *J(x)J(x)*J(x), находим новое приближение переменных xnew  с помощью следующей формулы:

$$x_{\text{new}} = x - J^{-1}(x) \cdot F(x)$$

где *J^(-1)(x)* — обратная матрица якобиана, а *F(x)F(x)*F(x) — вектор значений функций.

Чтобы найти xnew , нужно:

1. Вычислить матрицу *J(x)J(x)*J(x).
2. Найти её обратную (используем метод Гаусса или встроенный алгоритм).
3. Умножить на *F(x)F(x)*F(x) и вычесть результат из *xx*x.

```javascript
function newtonIteration(equations, initialGuess, tolerance = 1e-7, maxIterations = 100)
    let x = [...initialGuess];
    const jacobian = computeJacobian(equations, x);

    for (let iter = 0; iter < maxIterations; iter++) {
        const fx = evaluateSystem(equations, x);
        const jacobianInv = inverseMatrix(jacobian);

        // dx = -J^-1 * F(x)
        const dx = multiplyMatrixVector(jacobianInv, fx).map(value => -value);
        x = x.map((xi, i) => xi + dx[i]);

        // Check for convergence
        if (Math.sqrt(dx.reduce((sum, val) => sum + val * val, 0)) < tolerance) break;
    }
    return x;
}
```