```julia
stumpff_C2(z::Float64) = z > 0 ? (1 - cos(sqrt(z)))/z : z < 0 ? (cosh(sqrt(-z)) -
1)/(-z) : 1/2
stumpff_C3(z::Float64) = z > 0 ? (sqrt(z) - sin(sqrt(z))) / (z*sqrt(z)) : z < 0 ?
(sinh(sqrt(-z)) - sqrt(-z)) / ((-z)*sqrt(-z)) : 1/6

function solve_lambert(
        r1::Vector{Float64},
        r2::Vector{Float64},
        TOF::Float64;
        μ::Float64 = μ_Earth,
        long_way::Bool = false
)
        # Magnitudes
        r1_norm = norm(r1)
        r2_norm = norm(r2)
        # Transfer angle Δθ
        cos_dθ = dot(r1, r2) / (r1_norm * r2_norm)
        Δθ = acos(clamp(cos_dθ, -1.0, 1.0))
        if long_way
                Δθ = Δθ < π ? 2π - Δθ : Δθ
        else
                Δθ = Δθ > π ? 2π - Δθ : Δθ
        end
        # A-parameter
        A = sin(Δθ) * sqrt(r1_norm * r2_norm / (1 - cos(Δθ)))
        if iszero(A)
                error("Cannot compute Lambert solution: A = 0")
        end

        # Time-of-flight function F(z) = 0
        function F(z)
                C2 = stumpff_C2(z)
                C3 = stumpff_C3(z)
                y  = r1_norm + r2_norm + A * (z*C3 - 1) / sqrt(C2)
                if y < 0
                        return Inf
                end
                return ( (y/C2)^(3/2) * C3 + A*sqrt(y) ) / sqrt(μ) - TOF
        end

        # Solve for z via Newton-Raphson with finite-difference derivative
        z = 0.0
        for _ in 1:200
                Fz = F(z)
                if abs(Fz) < 1e-8
                        break
                end
                δ   = 1e-6
                dF  = (F(z + δ) - F(z - δ)) / (2δ)
                z  -= Fz / dF
        end

        # Compute y, f, g, g
        C2 = stumpff_C2(z)
        C3 = stumpff_C3(z)
        y  = r1_norm + r2_norm + A * (z*C3 - 1) / sqrt(C2)

        f    = 1 - y/r1_norm
        g    = A * sqrt(y/μ)
```

```
        gdot = 1 - y/r2_norm

        # Velocity vectors
        v1 = (r2 .- f*r1) ./ g
        v2 = (gdot*r2 .- r1) ./ g

        # Compute eccentricity and periapsis radius from (r1, v1)
        h_vec = cross(r1, v1)
        e_vec = (1/μ) * ((norm(v1)^2 - μ/r1_norm)*r1 .- dot(r1,v1)*v1)
        e     = norm(e_vec)
        # Semi-major axis from energy
        energy = norm(v1)^2/2 - μ/r1_norm
        a      = -μ / (2*energy)
        rp     = a * (1 - e)

        return v1, v2, e, rp
end

export solve_lambert
```