Sheet 5: Floating-point error (10 P)

Exercise 1

(taken from Higham 1.3)

The following expressions are cancellation-free or feature subtraction operations only for expressions involving the input arguments.

1. Use $\frac{x}{\sqrt{x+1}+1}$
2. Use $2\sin\left(\frac{x-y}{2}\right)\cos\left(\frac{x+y}{2}\right)$
3. Use $(x-y)(x+y)$
4. Use $\frac{\sin(x)}{1+\cos(x)}$ or $\tan(\frac{x}{2})$
5. Use

$$\begin{aligned} c &= \sqrt{a^2+b^2-2ab\cos(\theta)} \\ &= \sqrt{a^2+b^2-2ab(\cos(\theta)+1-1)} \\ &= \sqrt{a^2+b^2-2ab-2ab(\cos(\theta)-1)} \\ &= \sqrt{(a-b)^2+4ab\sin^2(\theta/2)} \end{aligned}$$

where we have used the half-angle identity $1-\cos(\theta) = 2\sin^2(\theta/2)$.

Exercise 2

(taken from Higham 2.5) We can use the geometric series

$$\sum_{i=1}^{\infty} r^i = \frac{1}{1-r} \quad \text{for } |r| < 1.$$

We have

$$\sum_{i=1}^{\infty} 2^{-4i} + 2^{-4i-1} = \frac{3}{2}\sum_{i=1}^{\infty}(2^{-4})^i = \frac{3}{2}\frac{2^{-4}}{1-2^{-4}} = \frac{3}{2}\frac{1}{15} = \frac{1}{10} = 0.1.$$

Since $x = 0.1$ in base 2 thus reads as

$$x = 0.1100\,1100\,1100\,1100\,1100\,1100\,|\,1100\,\ldots \times 2^{-3},$$

rounding to 24 bits yields

$$\hat{x} = 0.1100\,1100\,1100\,1100\,1100\,1101 \times 2^{-3}.$$

Thus

$$\begin{aligned}\hat{x} - x &= (0.001 - 0.000\overline{1100})_2 \times 2^{-21} \times 2^{-3} \\ &= \left(\frac{1}{8} - \frac{1}{10}\right) \times 2^{-24} \\ &= \frac{1}{40} \times 2^{-24},\end{aligned}$$

and conclude $(x - \hat{x})/x = -\frac{1}{4} \times 2^{-24} = -\frac{1}{4}u$

Exercise 3

(taken from Higham 2.22)

The problem with the simple implementation is the treatment of `NaN` as in particular `x > y` is `false` as soon as x or y are `NaN`. This breaks the symmetry in the arguments:

```
max_naive (generic function with 1 method)
```

```
function max_naive(a, b)
    if a > b
        return a
    else
        return b
    end
end
```

```
1.0
```

```
max_naive(NaN, 1.0)
```

```
NaN
```

```
max_naive(1.0, NaN)
```

An additional subtlety is that

```
false
```

```
0.0 > -0.0
```

or more precisely that `-0.0` and `0.0` are equal whereas we would like `max(-0.0, 0.0)` to yield the positive sign.

An alternative is to catch the `NaN` early and introduce a special branch for zero:

```
max_better (generic function with 1 method)
```

```
function max_better(a, b)
    if isnan(a)
        return a
    elseif isnan(b)
        return b
    elseif iszero(a) && iszero(b)
        return abs(a)
    elseif a > b
        return a
    else
        return b
    end
end
```

Testing some special cases:

-0.0

```
sign(-0.0)
```

0.0

```
max_better(0.0, -0.0)
```

NaN

```
max_better(1.0, NaN)
```

NaN

```
max_better(-NaN, 1.0)
```

Exercise 4

(a) The following points should be noticed by the students:

- The accuracy of the estimated eigenvalue in 16-bit arithmetic is very low. The eigenvector is of much higher accuracy, closer to what was requested via `tol`.
- This is a puzzeling artifact of reduced precision, since the eigenvalue should converge quadratically, hence be more accurate.
- As a result the residual norm (if computed in `Float64` arithmetic) is also much larger than `tol`. In contrast, the residual norm computed in **16**-bit arithmetic is two orders of magnitude smaller, closer to `tol` and gives away the impression that everything went fine. We cannot
- Overall we get a much lower quality eigenpair than what one would expect from the selected `tol` parameter and from the **16**-bit computed residual norm.
- We see that for ill-conditioned settings or reduced precision care should be taken and even a residual smaller than the desired accuracy —- if computed in a too low working precision —- might not be sufficient to ensure convergence.
- `BFloat16` versus `Float16` makes no real difference.

```
begin
    using LinearAlgebra
    using BFloat16s
    using IntervalArithmetic
end
```

```
power_method (generic function with 2 methods)
```

```
function power_method(A, u=randn(eltype(A), size(A, 2));
                      tol=1e-6, maxiter=100, verbose=true)
    norm_Δu = NaN
    for i in 1:maxiter
        u_prev = u
        u = A * u
        u /= norm(u)
        norm_Δu = min(norm(u - u_prev), norm(-u - u_prev))
        norm_Δu < tol && break
        verbose && println("$i   $norm_Δu")
    end
    μ = dot(u, A, u)
    norm_Δu ≥ tol && verbose && @warn "Power not converged $norm_Δu"
    (; μ, u)
end
```

```
A = 3×3 Matrix{Float64}:
 -9.99922      0.000914606  -0.000398113
 -0.00077258  30.001        -0.0018039
  0.00138971   0.001402      0.199134
```

```
A = diagm([-10, 30, 0.2]) + 1e-3 * randn(3, 3)
```

```
Eigen{Float64, Float64, Matrix{Float64}, Vector{Float64}}
values:
3-element Vector{Float64}:
 -9.999219615171121
  0.19913384325140607
 30.001021161532304
vectors:
3×3 Matrix{Float64}:
  1.0          -3.90316e-5  2.28645e-5
  1.93082e-5    6.05288e-5  1.0
 -0.000136271   1.0         4.70451e-5
```

```
begin
    λ, X = eigen(A)
    μ_ref = λ[end]
    u_ref = X[:, end]
    Eigen(λ, X)
end
```

```
let
    (;μ, u) = power_method(Float16.(A); tol=1e-6)
    α = sign(dot(u, u_ref))

    @show abs(μ - μ_ref)       # Exact only to 1e-3
    @show norm(u - α * u_ref)  # Exact to 1e-7
    @show norm(Float16.(A) * u - μ * u)  # Residual computed in working precision
    @show norm(A * u - μ * u)  # 2 orders larger than the one above!
end;
```

```
1   1.471
2   1.397
3   0.7144
4   0.2605
5   0.08777
6   0.0293
7   0.009766
8   0.003258
9   0.001092
10   0.0003452
11   0.0001206
12   4.02e-5
13   1.34e-5
14   4.5e-6
15   1.43e-6
abs(μ - μ_ref) = 0.0010211615323036938
norm(u - α * u_ref) = 1.438619030491379e-7
norm(Float16.(A) * u - μ * u) = Float16(5.8e-6)
norm(A * u - μ * u) = 0.0010211790715605375
```

```
let
    (;μ, u) = power_method(BFloat16.(A); tol=1e-6)
    α = sign(dot(u, u_ref))

    @show abs(μ - μ_ref)       # Exact only to 1e-3
    @show norm(u - α * u_ref)  # Exact to 1e-7
    @show norm(BFloat16.(A) * u - μ * u)  # Residual computed in working precision
    @show norm(A * u - μ * u)  # 2 orders larger than the one above!
end;
```

```
1   BFloat16(2.703125)
2   BFloat16(0.1953125)
3   BFloat16(0.06542969)
4   BFloat16(0.021850586)
5   BFloat16(0.007293701)
6   BFloat16(0.0024414062)
7   BFloat16(0.0008163452)
8   BFloat16(0.00027275085)
9   BFloat16(9.10759f-5)
10   BFloat16(3.0398369f-5)
11   BFloat16(1.013279f-5)
12   BFloat16(3.2186508f-6)
abs(μ - μ_ref) = 0.0010211615323036938
norm(u - α * u_ref) = 2.2805142865239888e-7
norm(BFloat16.(A) * u - μ * u) = BFloat16(7.6293945f-6)
norm(A * u - μ * u) = 0.0010212023076042547
```

(b) Employing e.g. `tol=1e-6` is sufficient to get the upper bound of the interval resulting in the residual computation below 10^{-5}, thus we have achieved the goal of the problem. Proving a residual norm much below 10^{-6} cannot be achieved.

```
let
    (;μ, u) = power_method(Float32.(A); tol=1e-10)
    @show norm(Float32.(A) * u - μ * u)  # Computation in Float32

    # Computation using intervals:
    @show norm(
        interval.(A) * interval.(u) - interval(μ) * interval.(u)
    )
end;
```

```
1   0.8508705
2   1.2909694
3   0.5920468
4   0.20956708
5   0.0703634
6   0.023471165
7   0.007823569
8   0.0026075903
9   0.00086910045
10   0.00028966772
11   9.654509e-5
12   3.217809e-5
13   1.0724828e-5
14   3.5745434e-6
15   1.1913817e-6
16   3.9708175e-7
17   1.3234603e-7
18   4.4112312e-8
19   1.4702891e-8
20   4.9003575e-9
21   1.6334525e-9
22   5.4387783e-10
23   1.8189894e-10
norm(Float32.(A) * u - μ * u) = 5.820766f-10
norm(interval.(A) * interval.(u) - interval(μ) * interval.(u)) = Interval{Floa
t64}(7.300139684475402e-7, 7.300140308399387e-7)
```