MATH50003 Combined Problem Sheet Solutions

Original Author: Dr. Sheehan Olver Compiled by: Isaac Lee | **Date: 16/03/2022** | Github Source

MATH50003 Problem Sheet 1

This problem sheet tests the representation of numbers on the computer, using modular arithmetic. We also use floating point rounding modes to implement interval arithmetic, and thereby
produce rigorous bounds on the exponential.

```
using ColorBitstring, SetRounding
```

Questions marked with a ⋆ are meant to be completed without using a computer.
Problems are denoted A/B/C to indicate their difficulty.

1. Binary representation

Problem 1.1 (C) What is the binary representation of $1/5$? (Hint: use `printbits` to derive a proposed form.)

SOLUTION
Note that

```
printbits(1/5)
```

Hence we show that

$$(0.00110011001100\ldots)_2 = (2^{-3}+2^{-4})(1.00010001000\ldots)_2 = (2^{-3}+2^{-4})\sum_{k=0}^{\infty}\frac{1}{16^k}$$
$$= \frac{2^{-3}+2^{-4}}{1-\frac{1}{2^4}} = \frac{3}{15} = \frac{1}{5}$$

Problem 1.2 (C) What is π to 5 binary places? Hint: recall that $\pi \approx 3.14$.

SOLUTION
Note that

```
3 + 1/8  + 1/64
```

which has the binary representation $(11.001001)_2$.
Indeed:

```
printbits(Float16(π))
```

Instead of simply guessing the above representation we can instead continuously subtract the largest powers 2 which do not result in a negative number. For π the procedure then finds that we can write $\pi - 1*2^1 - 1*2^0 - 1*2^{-3} - 1*2^{-6}\ldots$

2. Integers

Problem 2.1 (C) With 8-bit signed integers, find the bits for the following: $10, 120, -10$.

SOLUTION

We can find the binary digits by repeatedly subtracting the largest power of 2 less than a number until we reach 0, e.g. $10 - 2^3 - 2 = 0$
implies $10 = (1010)_2$.
Thus the bits are:

```
printlnbits(Int8(10))
```

Similarly,

$$120 = 2^6 + 2^5 + 2^4 + 2^3 = (1111000)_2$$

Thus the bits are:

```
printlnbits(Int8(120))
```

For negative numbers we perform the same trick but adding 2^p to make it positive, e.g.,

$$-10 = 2^8 - 10 (\mathrm{mod} 2^8) = 246 = 2^7 + 2^6 + 2^5 + 2^4 + 2^2 + 2 = (11110110)_2$$

This the bits are:

```
printlnbits(Int8(-10))
```

Problem 2.2 (C) What will `Int8(120) + Int8(10)` return?

SOLUTION

It will return

$$130 (\mathrm{mod}\ 2^8) = -126 (\mathrm{mod}\ 2^8)$$

```
Int8(120) + Int8(10)
```

3. Floating point numbers

Problem 3.1 (C) What are the single precision F_{32} (`Float32`) floating point representations for the following:

$$2, 31, 32, 23/4, (23/4) \times 2^{100}$$

Check your answers using `printbits`.

SOLUTION

Recall that we have `σ,Q,S = 127,8,23`. Thus we write

$$2 = 2^{128-127} * (1.00000000000000000000000)_2$$

The exponent bits are those of

$$128 = 2^7 = (10000000)_2$$

Hence we get

```
printlnbits(2f0)
```

We write

$$31 = (11111)_2 = 2^{131-127} * (1.1111)_2$$

And note that $131 = (10000011)_2$ Hence we have:

```
printlnbits(31f0)
```

On the other hand,

$$32 = (100000)_2 = 2^{132-127}$$

and $132 = (10000100)_2$ hence:

```
printlnbits(32f0)
```

Note that

$$23/4 = 2^{-2} * (10111)_2 = 2^{129-127} * (1.0111)_2$$

and $129 = (10000001)_2$ hence we get:

```
printlnbits(23f0/4)
```

Finally,

$$23/4 * 2^{100} = 2^{229-127} * (1.0111)_2$$

and $229 = (11100101)_2$ giving us:

```
printlnbits(23f0/4 * 2f0^100)
```

Problem 3.2 (B) Let $m(y) = \min\{x \in F_{32} : x > y\}$ be the smallest single precision number greater than y. What is $m(2) - 2$ and $m(1024) - 1024$? Check your answer using the **`nextfloat`** command.

SOLUTION

The next float after 2 is $2 * (1 + 2^{-23})$ hence we get $m(2) - 2 = 2^{-22}$:

```
nextfloat(2f0) - 2, 2^(-22)
```

similarly, for $1024 = 2^{10}$ we find that the difference $m(1024) - 1024$ is $2^{10-23} = 2^{-13}$:

```
nextfloat(1024f0) - 1024, 2^(-13)
```

4. Arithmetic

Problem 4.1 (C) Suppose $x = 1.25$ and consider 16-bit floating point arithmetic (`Float16`).
What is the error in approximating x by the nearest float point number $\mathrm{fl}(x)$?
What is the error in approximating $2x$, $x/2$, $x + 2$ and $x - 2$ by $2 \otimes x$, $x \oslash 2$, $x \oplus 2$ and $x \ominus 2$?

SOLUTION
None of these computations have errors since they are all exactly representable as floating point numbers.

Problem 4.2 (B) For what floating point numbers is $x \oslash 2 \neq x/2$ and $x \oplus 2 \neq x + 2$?

SOLUTION

Consider a normal $x = 2^{q-\sigma}(1.b_1 \dots b_S)_2$.
Provided $q > 1$ we have

$$x \oslash 2 = x/2 = 2^{q-\sigma-1}(1.b_1 \dots b_S)_2$$

However, if $q = 1$ we lose a bit as we shift:

$$x \oslash 2 = 2^{1-\sigma}(0.b_1 \dots b_{S-1})_2$$

and the property will be satisfy if $b_S = 1$.
Similarily if we are sub-normal, $x = 2^{1-\sigma}(0.b_1 \dots b_S)_2$ and
we have

$$x \oslash 2 = 2^{1-\sigma}(0.0b_1 \dots b_{S-1})_2$$

and the property will be satisfy if $b_S = 1$.
(Or `NaN`.)

Here are two examples:

```
# normal number with q = 1 and last bit 1
x = reinterpret(Float16, parse(UInt16, "0000010000000011"; base=2))
x/2 == Float64(x)/2 # Float64 can exactly represent x/2
```

```
# sub-normal number with q = 1 and last bit 1
x = reinterpret(Float16, parse(UInt16, "0000000000000011"; base=2))
x/2 == Float64(x)/2 # Float64 can exactly represent x/2
```

For the second part, Similar to the next problem,
we see that the property holds true if $|x| < 2^{S+2} - 1$,
as otherwise:

```
x = Float16(2)^(12)-1 # bits 0110110000000000
x+2 == x
```

We see this is sharp:

```
y = prevfloat(x)
y+2 == y
```

Problem 4.3 (A) Explain why the following return **`true`**. What is the largest floating point number **`y`** such that **`y + 1 ≠ y`**?

```
x = 10.0^100
x + 1 == x
```

SOLUTION

Writing $10 = 2^3(1.01)_2$ we have

$$\mathrm{fl}(10^{100}) = \mathrm{fl}(2^{300}(1+2^{-4})^{100}) = 2^{300}(1.\mathrm{b}_1\dots\mathrm{b}_{52})_2$$

where the bits b_k are not relevant. We then have:

$$\mathrm{fl}(10^{100}) \oplus 1 = \mathrm{fl}(2^{300}[(1.\mathrm{b}_1\dots\mathrm{b}_{52})_2 + 2^{-300}]) = \mathrm{fl}(10^{100})$$

since 2^{-300} is below the necessary precision.

The largest floating point number satisfying the condition is $y = 2^{53} - 1$, since
$S = 52$. First note 2^{53} does not satisfy the condition:

```
x = 2.0^53
x + 1 == x
```

We can however successfully create the previsous float $2^{53} - 1$ by subtracting
(Explain why this works while **`x+1`** fails):

```
y = x - 1
printlnbits(x)
printlnbits(y)
```

And this satisfies:

```
y + 1 ≠ y
```

Problem 4.4 (B) What are the exact bits for $1/5, 1/5 + 1$ computed using half-precision arithmetic (`Float16`) (using default rounding)?

SOLUTION

We saw above that

$$1/5 = 2^{-3} * (1.10011001100\ldots)_2 \approx 2^{-3} * (1.1001100110)_2$$

where the $\approx$ is rounded to the nearest 10 bits (in this case rounded down).
We write $-3 = 12 - 15$
hence we have $q = 12 = (01100)_2$.
so we get the bits:

```
printbits(Float16(1)/5)
```

Adding `1` we get:

$$1 + 2^{-3} * (1.1001100110)_2 = (1.001100110011)_2 \approx (1.0011001101)_2$$

Here we write the exponent as $0 = 15 - 15$ where $q = 15 = (01111)_2$.
Thus we get:

```
printbits(1 + Float16(1)/5)
```

Problem 4.5 (A) Explain why the following does not return `1`. Can you compute the bits explicitly?

```
Float16(0.1) / (Float16(1.1) - 1)
```

SOLUTION
For the last problem, note that

$$\frac{1}{10} = \frac{1}{2}\frac{1}{5} = 2^{-4} * (1.10011001100\ldots)_2$$

hence we have

$$\mathrm{fl}(\frac{1}{10}) = 2^{-4} * (1.1001100110)_2$$

and

$$\mathrm{fl}(1 + \frac{1}{10}) = \mathrm{fl}(1.0001100110011\ldots) = (1.0001100110)_2$$

Thus

$$\mathrm{fl}(1.1) \ominus 1 = (0.0001100110)_2 = 2^{-4}(1.1001100000)_2$$

and hence we get

$$\mathrm{fl}(0.1) \oslash (\mathrm{fl}(1.1) \ominus 1) = \mathrm{fl}(\frac{(1.1001100110)_2}{(1.1001100000)_2}) \neq 1$$

To compute the bits explicitly, write $y = (1.10011)_2$ and divide through to get:

$$\frac{(1.1001100110)_2}{(1.10011)_2} = 1 + \frac{2^{-8}}{y} + \frac{2^{-9}}{y}$$

We then have

$$y^{-1} = \frac{32}{51} = 0.627\ldots = (0.101\ldots)_2$$

Hence

$$1 + \frac{2^{-8}}{y} + \frac{2^{-9}}{y} = 1 + (2^{-9} + 2^{-11} + \cdots) + (2^{-10} + \cdots) = (1.00000000111\ldots)_2$$

Therefore we round up (the `...` is not exactly zero but if it was it would be a tie and we would round up anyways to get a zero last bit) and get:

```
printlnbits(Float16(0.1) / (Float16(1.1) - 1))
```

Problem 4.6 (B) Find a bound on the *absolute error* in terms of a constant times ϵ_{m} for the following computations

$$(1.1 * 1.2) + 1.3$$
$$(1.1 - 1)/0.1$$

implemented using floating point arithmetic (with any precision).

SOLUTION

The first problem is very similar to what we saw in lecture. Write

$$(\mathrm{fl}(1.1) \otimes \mathrm{fl}(1.2)) \oplus \mathrm{fl}(1.3) = (1.1(1+\delta_1)1.2(1+\delta_2)(1+\delta_3) + 1.3(1+\delta_4))(1+\delta_5)$$

We first write

$$1.1(1+\delta_1)1.2(1+\delta_2)(1+\delta_3) = 1.32(1+\delta_6)$$

where

$$|\delta_6| \leq |\delta_1| + |\delta_2| + |\delta_3| + |\delta_1||\delta_2| + |\delta_1||\delta_3| + |\delta_2||\delta_3| + |\delta_1||\delta_2||\delta_3| \leq 4\epsilon_{\mathrm{m}}$$

Then we have

$$1.32(1+\delta_6) + 1.3(1+\delta_4) = 2.62 + \underbrace{1.32\delta_6 + 1.3\delta_4}_{\delta_7}$$

where

$$|\delta_7| \leq 7\epsilon_{\mathrm{m}}$$

Finally,

$$(2.62 + \delta_6)(1 + \delta_5) = 2.62 + \underbrace{\delta_6 + 2.62\delta_5 + \delta_6\delta_5}_{\delta_8}$$

where

$$|\delta_8| \leq 10\epsilon_{\mathrm{m}}$$

For the second part, we do:

$$(\mathrm{fl}(1.1) \ominus 1) \oslash \mathrm{fl}(0.1) = \frac{(1.1(1 + \delta_1) - 1)(1 + \delta_2)}{0.1(1 + \delta_3)}(1 + \delta_4)$$

Write

$$\frac{1}{1 + \delta_3} = 1 + \delta_5$$

where

$$|\delta_5| \leq \left| \frac{\delta_3}{1 + \delta_3} \right| \leq \frac{\epsilon_{\mathrm{m}}}{2} \frac{1}{1 - 1/2} \leq \epsilon_{\mathrm{m}}$$

using the fact that $|\delta_3| < 1/2$.
Further write

$$(1 + \delta_5)(1 + \delta_4) = 1 + \delta_6$$

where

$$|\delta_6| \leq |\delta_5| + |\delta_4| + |\delta_5||\delta_4| \leq 2\epsilon_{\mathrm{m}}$$

We also write

$$\frac{(1.1(1 + \delta_1) - 1)(1 + \delta_2)}{0.1} = 1 + \underbrace{11\delta_1 + \delta_2 + 11\delta_1\delta_2}_{\delta_7}$$

where

$$|\delta_7| \leq 17\epsilon_{\mathrm{m}}$$

Then we get

$$(\mathrm{fl}(1.1) \ominus 1) \oslash \mathrm{fl}(0.1) = (1 + \delta_7)(1 + \delta_6) = 1 + \delta_7 + \delta_6 + \delta_6\delta_7$$

and the error is bounded by:

$$(17 + 2 + 34)\epsilon_{\mathrm{m}} = 53\epsilon_{\mathrm{m}}$$

This is quite pessimistic but still captures that we are on the order of ϵ_{m}.

5. Interval arithmetic

The following problems consider implementation of interval arithmetic for proving precise bounds on arithmetic operations. That is recall the set operations

$$A + B = \{x + y : x \in A, y \in B\}, AB = \{xy : x \in A, y \in B\}.$$

Problem 5.1 (B) For intervals $A = [a, b]$ and $B = [c, d]$ such that $0 \notin A, B$ and integer $n \neq 0$,
deduce formulas for the minimum and maximum of A/n, $A + B$ and AB.

Solution

$$\begin{aligned}
\frac{A}{n} &= \begin{cases} [a/n, b/n] & n > 0 \\ [b/n, a/n] & n < 0' \end{cases} \\
A + B &= [a + c, b + d] \\
AB &= \begin{cases} [bd, ac] & a, b, c, d < 0 \\ [ad, bc] & a, b < 0 \text{ and } c, d > 0 \\ [bc, ad] & a, b > 0 \text{ and } c, d < 0 \\ [ac, bd] & a, b, c, d > 0 \end{cases}
\end{aligned}$$

Problem 5.2 (B)
We want to implement floating point variants such that, for $S = [a, b] + [c, d]$ $P = [a, b] * [c, d]$, and $D = [a, b]/n$ for an integer n,

$$\begin{aligned}
[a, b] \oplus [c, d] &:= [\mathrm{fl}^{\mathrm{down}}(\min S), \mathrm{fl}^{\mathrm{up}}(\max S)] \\
[a, b] \otimes [c, d] &:= [\mathrm{fl}^{\mathrm{down}}(\min P), \mathrm{fl}^{\mathrm{up}}(\max P)] \\
[a, b] \oslash n &:= [\mathrm{fl}^{\mathrm{down}}(\min D), \mathrm{fl}^{\mathrm{up}}(\max D)]
\end{aligned}$$

This guarantees $S \subseteq [a, b] \oplus [c, d]$, $P \subseteq [a, b] \otimes [c, d]$, and $D \subseteq [a, b] \oslash n$.
In other words, if $x \in [a, b]$ and $y \in [c, d]$ then $x + y \in [a, b] \oplus [c, d]$, and we thereby have bounds on $x + y$.

Use the formulae from Problem 5.1 to complete (by replacing the `# TODO: …` comments with code) the following implementation of an `Interval` so that `+`, `-`, and `/` implement $\oplus$, $\ominus$, and $\oslash$ as defined above.

```
# Interval(a,b) represents the closed interval [a,b]
struct Interval{T}
   a::T
   b::T
end

import Base: *, +, -, /, one, in

# create an interval corresponding to [1,1]
one(x::Interval) = Interval(one(x.a), one(x.b))

# Support x in Interval(a,b)
in(x, y::Interval) = y.a ≤ x ≤ y.b

# Following should implement ⊕
function +(x::Interval, y::Interval)
```

```
    T = promote_type(typeof(x.a), typeof(x.b) )
    a = setrounding(T, RoundDown) do
        # TODO: upper bound
        ## SOLUTION
        x.a + y.a
        ## END
    end
    b = setrounding(T, RoundUp) do
        # TODO: upper bound
        ## SOLUTION
        x.b + y.b
        ## END
    end
    Interval(a, b)
end

# Following should implement ⊘
function /(x::Interval, n::Integer)
    T = typeof(x.a)
    if iszero(n)
        error("Dividing by zero not support")
    end
    a = setrounding(T, RoundDown) do
        # TODO: lower bound
        ## SOLUTION
        if n > 0
            x.a / n
        else
            x.b / n
        end
        ## END
    end
    b = setrounding(T, RoundUp) do
        # TODO: upper bound
        ## SOLUTION
        if n > 0
            x.b / n
        else
            x.a / n
        end
        ## END
    end
    Interval(a, b)
end

# Following should implement ⊗
function *(x::Interval, y::Interval)
    T = promote_type(typeof(x.a), typeof(x.b))
    if 0 in x || 0 in y
        error("Multiplying with intervals containing 0 not supported.")
    end
    a = setrounding(T, RoundDown) do
        # TODO: lower bound
        ## SOLUTION
        if x.a < 0 && x.b < 0 && y.a < 0 && y.b < 0
            y.b * x.b
```

```
        elseif x.a < 0 && x.b < 0 && y.a > 0 && y.b > 0
            x.a * y.b
        elseif x.a > 0 && x.b > 0 && y.a < 0 && y.b < 0
            x.b * y.a
        else
            x.a * y.a
        end
        ## END
    end
    b = setrounding(T, RoundUp) do
        # TODO: upper bound
        ## SOLUTION
        if x.a < 0 && x.b < 0 && y.a < 0 && y.b < 0
            y.a * x.a
        elseif x.a < 0 && x.b < 0 && y.a > 0 && y.b > 0
            x.b * y.a
        elseif x.a > 0 && x.b > 0 && y.a < 0 && y.b < 0
            x.a * y.b
        else
            x.b * y.b
        end
        ## END
    end
    Interval(a, b)
end
```

Problem 5.3 (A) The following function computes the first `n+1` terms of the Taylor series of $\exp(x)$:

$$\sum_{k=0}^{n} \frac{x^k}{k!}$$

```
function exp_t(x, n)
    ret = one(x) # 1 of same type as x
    s = one(x)
    for k = 1:n
        s = s/k * x
        ret = ret + s
    end
    ret
end
```

Bound the tail of the Taylor series for e^x assuming $|x| \leq 1$.
(Hint: $\mathrm{e}^x \leq 3$ for $x \leq 1$.)
Use the bound
to complete the function `exp_bound` which computes e^x with rigorous error bounds, that is
so that when applied to an interval $[a, b]$ it returns an interval that is
guaranteed to contain the interval $[\mathrm{e}^a, \mathrm{e}^b]$.

```
function exp_bound(x::Interval, n)
    if abs(x.a) > 1 || abs(x.b) > 1
        error("Interval must be a subset of [-1, 1]")
    end
```

```
    ret = exp_t(x, n) # the code for Taylor series should work on Interval unmodified
    f = factorial(min(20, n + 1)) # avoid overflow in computing factorial
    T = typeof(ret.a)

    # TODO: modify ret so that exp(x) is guaranteed to lie in it
    ## SOLUTION
    err = setrounding(T, RoundUp) do
        3 / f
    end
    ret + Interval(-err,err)
    ## END
end
```

Check your result for computing e and e^{-1} by assuring that
the following returns true:

```
exp(big(1)) in exp_bound(Interval(1.0,1.0), 20) && exp(big(-1)) in exp_bound(Interval(-1.0,
```

Further, ensure that the width of each returned interval is less than 10^{-14}.

SOLUTION From the Taylor remainder theorem we know the error is

$$\frac{f^{(k+1)}(\xi)}{(k+1)!}|x|^{k+1} \leq \frac{3}{(k+1)!}$$

Thus by widening the computation by this error we ensure that we have
captured the error by truncating the Taylor series.

MATH50003 Numerical Analysis: Problem Sheet 2

This week we look at other variants of finite-differences, including central differences and second-order finite-differences. We also investigate mathematical properties of dual numbers and extend their implementation to
other functions. Finally, we see how dual numbers can be combined with Newton iteration for root finding.

Questions marked with a ⋆ are meant to be completed without using a computer.
Problems are denoted A/B/C to indicate their difficulty.

1. Finite-differences

Problem 1.1⋆ (B) Use Taylor's theorem to derive an error bound for central differences

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}.$$

Find an error bound when implemented in floating point arithmetic, assuming that
$f^{\mathrm{FP}}(x) = f(x) + \delta_x$
where $|\delta_x| \leq c\epsilon_{\mathrm{m}}$.

SOLUTION

By Taylor's theorem, the approximation around $x+h$ is
$f(x+h) = f(x) + f'(x)h + \frac{f''(x)}{2}h^2 + \frac{f'''(z_1)}{6}h^3,$
for some $z_1 \in (x, x+h)$ and similarly
$f(x-h) = f(x) + f'(x)(-h) + \frac{f''(x)}{2}h^2 - \frac{f'''(z_2)}{6}h^3,$
for some $z_2 \in (x-h, x)$.

Subtracting the second expression from the first we obtain
$f(x+h) - f(x-h) = f'(x)(2h) + \frac{f'''(z_1)+f'''(z_2)}{6}h^3.$
Hence,

$$\frac{f(x+h)-f(x-h)}{2h} = f'(x) + \underbrace{\frac{f'''(z_1)+f'''(z_2)}{12}h^2}_{\delta_{\text{Taylor}}}.$$

Thus, the error can be bounded by
$|\delta_{\text{Taylor}}| \le \frac{M}{6}h^2,$
where

$$M = \max_{y\in[x-h,x+h]} |f'''(y)|.$$

In floating point we have

$$\begin{aligned}(f^{\text{FP}}(x+2h) \ominus f^{\text{FP}}(x-2h)) \oslash (2h) &= \frac{f(x+h)+\delta_{x+h} - f(x-h) - \delta_{x-h}}{2h}(1+\delta_1) \\ &= \frac{f(x+h)-f(x-h)}{2h}(1+\delta_1) + \frac{\delta_{x+h}-\delta_{x-h}}{2h}(1+\delta_1)\end{aligned}$$

Applying Taylor's theorem we get

$$(f^{\text{FP}}(x+h) \ominus f^{\text{FP}}(x-h)) \oslash (2h) = f'(x) + \underbrace{f'(x)\delta_1 + \delta_{\text{Taylor}}(1+\delta_1) + \frac{\delta_{x+h}-\delta_{x-h}}{2h}(1+\delta_1)}_{\delta^{\text{CD}}_{x,h}}$$

where

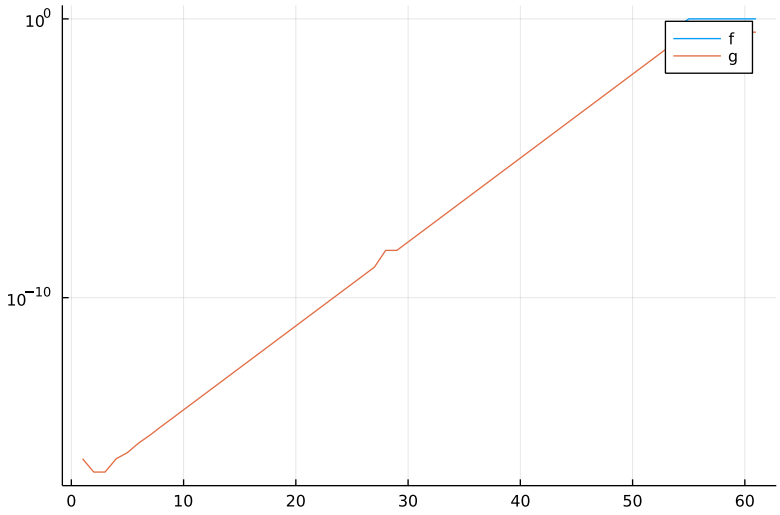
$$|\delta^{\text{CD}}_{x,h}| \le \frac{|f'(x)|}{2}\epsilon_{\text{m}} + \frac{M}{3}h^2 + \frac{2c\epsilon_{\text{m}}}{h}$$

Problem 1.2 (B) Implement central differences for $f(x) = 1 + x + x^2$ and $g(x) = 1 + x/3 + x^2$.
Plot the errors for **`h = 2.0 .^ (0:-1:-60)`** and **`h = 10.0 .^ (0:-1:-16)`**.
Derive the error exactly for the different cases to explain the observed behaviour.

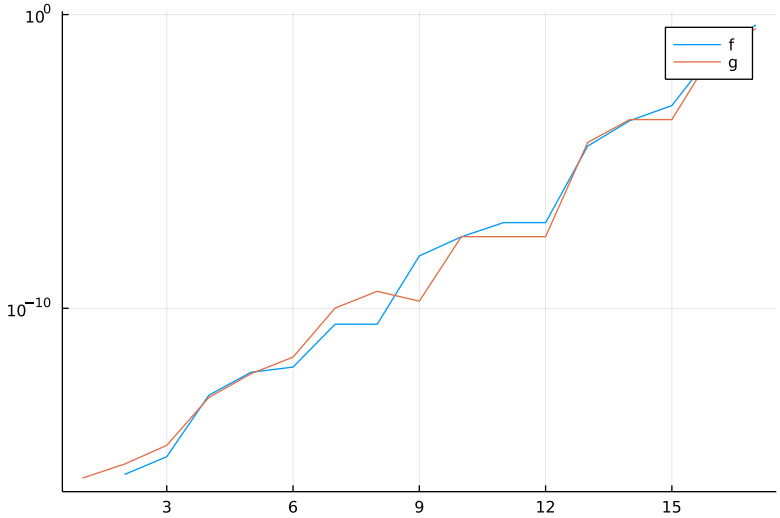
SOLUTION

```
using Plots

# define the functions
f = x -> 1 + x + x^2
g = x -> 1 + x/3 + x^2

# define analytic first derivatives for comparison
fp  = x -> 1 + 2 *x
gp = x ->1/3 + 2 *x

# central difference derivative approximation
centraldiff(x, h, f) = (f(x + h) - f(x - h))/(2 *h)
```

```
# computes an error
centraldifferror(x, h, f, fp) = abs(centraldiff(x, h, f) - fp(x))


#plotting f and g errors
x = 0.0 # some arbitrary point

# helper function to avoid trying to take logs of 0 in plots
nanabs(x) = iszero(x) ? NaN : abs(x)

# We find the error for the derivative of f is 0
# (until we run into the errors for too small h we discussed in the lecture)
h = 2.0 .^ (0:-1:-60)
plot(nanabs.(centraldifferror.(x, h, f, fp)), yaxis=:log10, label="f")
plot!(nanabs.(centraldifferror.(x, h, g, gp)), yaxis=:log10, label="g")
```




```
h = 10.0 .^ (0:-1:-16)
plot(nanabs.(centraldifferror.(x, h, f, fp)), yaxis=:log10, label="f")
plot!(nanabs.(centraldifferror.(x, h, g, gp)), yaxis=:log10, label="g")
```



To compute the errors of the central difference approximation of $f'(x)$ we compute

$$\left|\frac{f(x+h)-f(x-h)}{2h} - f'(x)\right| =$$
$$= \left|\frac{1+(x+h)+(x+h)^2-1-(x-h)-(x-h)^2}{2h} - (1+2x)\right| =$$
$$= \left|\frac{2h+4hx}{2h} - 1 - 2x\right| = 0.$$

As we can see, in this case the central difference approximation is exact. The errors we start observing for small step sizes are thus numerical in nature. The values of the function at $f(x+h)$ and $f(x-h)$ eventually become numerically indistinguishable and thus this finite difference approximation to the derivative incorrectly results in 0.

To compute the errors of the central difference approximation of $g'(x)$ we compute

$$\left|\frac{g(x+h)-g(x-h)}{2h} - g'(x)\right| =$$
$$= \left|\frac{1+\frac{(x+h)}{3}+(x+h)^2-1-\frac{(x-h)}{3}-(x-h)^2}{2h} - \left(\frac{1}{3}+2x\right)\right| =$$
$$= \left|\frac{1}{3} + 2x - \frac{1}{3} - 2x\right| = 0.$$

Problem 1.3 (A)⋆ Use Taylor's theorem to derive an error bound on the second-order derivative approximation

$$f''(x) \approx \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}$$

Find an error bound when implemented in floating point arithmetic, assuming that

$$f^{\mathrm{FP}}(x) = f(x) + \delta_x$$

where $|\delta_x| \leq c\epsilon_{\mathrm{m}}$.

SOLUTION

Using the same two formulas as in 1.1 we have
$f(x+h) = f(x) + f'(x)h + \frac{f''(x)}{2}h^2 + \frac{f'''(z_1)}{6}h^3,$
for some $z_1 \in (x, x+h)$
and
$f(x-h) = f(x) + f'(x)(-h) + \frac{f''(x)}{2}h^2 - \frac{f'''(z_2)}{6}h^3,$
for some $z_2 \in (x-h, x)$.

Summing the two we obtain
$f(x+h) + f(x-h) = 2f(x) + f''(x)h^2 + \frac{f'''(z_1)}{6}h^3 - \frac{f'''(z_2)}{6}h^3.$

Thus,
$f''(x) = \frac{f(x+h)-2f(x)+f(x-h)}{h^2} + \frac{f'''(z_2)-f'''(z_1)}{6}h.$

Hence, the error is
$\left|f''(x) - \frac{f(x+h)-2f(x)+f(x-h)}{h^2}\right| = \left|\frac{f'''(z_2)-f'''(z_1)}{6}h\right| \leq 2Ch,$
where again
$C = \max_{y\in[x-h,x+h]}\left|\frac{f'''(y)}{6}\right|.$

In floating point arithmetic, the error is
$\left|f''^{FP}(x) - f''(x)\right| = \left|\frac{f^{FP}(x+h)-2f^{FP}(x)+f^{FP}(x-h)}{h^2} - \frac{f(x+h)-2f(x)+f(x-h)}{h^2} - \frac{f'''(z_2)-f'''(z_1)}{6}h\right|$
$\leq \left|\frac{(f^{FP}(x+h)-f(x+h))-2(f^{FP}(x)-f(x))+(f^{FP}(x-h)-f(x-h))}{h^2}\right| + \left|\frac{f'''(z_2)-f'''(z_1)}{6}h\right|$
$\leq \left|\frac{\delta_{x+h}-2\delta_x+\delta_{x-h}}{h^2}\right| + 2Ch \leq \frac{4c\epsilon_m}{h^2} + 2Ch.$

Problem 1.4 (B) Use finite-differences, central differences, and second-order finite-differences to approximate to 5-digits the first and second
derivatives to the following functions
at the point $x = 0.1$:

$$\exp\left(\exp x \cos x + \sin x\right), \prod_{k=1}^{1000}\left(\frac{x}{k} - 1\right), \text{ and } f^{\mathrm{s}}_{1000}(x)$$

where $f^{\mathrm{s}}_n(x)$ corresponds to n-terms of the following continued fraction:

$$1 + \cfrac{x-1}{2 + \cfrac{x-1}{2 + \cfrac{x-1}{2 + \ddots}}},$$

e.g.:
$f^{\mathrm{s}}_1(x) = 1 + \frac{x-1}{2}$
$f^{\mathrm{s}}_2(x) = 1 + \frac{x-1}{2+\frac{x-1}{2}}$
$f^{\mathrm{s}}_3(x) = 1 + \frac{x-1}{2+\frac{x-1}{2+\frac{x-1}{2}}}$

SOLUTION

```
# Forward Difference
forwarddiff(x, h, f) = (f(x + h) - f(x))/h

# We already implemented central differences in a previous problem

# Second derivative via finite difference
finitediffsecond(x, h, f) = (f(x + h) - 2 * f(x) + f(x - h))/ (h^2)

# Define the functions
f = x -> exp(exp(x)cos(x) + sin(x))
g = x -> prod([x] ./ (1:1000) .- 1)
function cont(n, x)
   ret = 2*one(x)
   for k = 1:n-1
       ret = 2 + (x-1)/ret
   end
   1 + (x-1)/ret
end

# Choose our point
x = 0.1;
```

```
# We have to be a bit careful with the choice of h
h = eps()
# Values for exp(exp(x)cos(x) + sin(x))
println("f'($x) with forward difference: ", forwarddiff(x, sqrt(h), f))
println("f'($x) with central difference: ", centraldiff(x, cbrt(h), f))
println("f''($x) via finite difference: ", finitediffsecond(x, cbrt(h), f))
```

```
# Values for prod([x] ./ (1:1000) .- 1)
println("f'($x) with forward difference: ", forwarddiff(x, sqrt(h), g))
println("f'($x) with central difference: ", centraldiff(x, cbrt(h), g))
println("f''($x) via finite difference: ", finitediffsecond(x, cbrt(h), g))
```

```
# Values for the continued fraction
println("f'($x) with forward difference: ", forwarddiff(x, sqrt(h), x->cont(1000,x)))
println("f'($x) with central difference: ", centraldiff(x, sqrt(h), x->cont(1000,x)))
println("f''($x) via finite difference: ", finitediffsecond(x, cbrt(h), x->cont(1000,x)))
```

2. Dual numbers

Problem 2.1⋆ (C)

Show that dual numbers $\mathbb{D}$ are a *commutative ring*, that is, for all $a, b, c \in \mathbb{D}$ the following are satisfied:

1. *additive associativity*: $(a + b) + c = a + (b + c)$
2. *additive commutativity*: $a + b = b + a$
3. *additive identity*: There exists $0 \in \mathbb{D}$ such that $a + 0 = a$.
4. *additive inverse*: There exists $-a$ such that $(-a) + a = 0$.
5. *multiplicative associativity*: $(ab)c = a(bc)$
6. *multiplictive commutativity*: $ab = ba$
7. *multiplictive identity*: There exists $1 \in \mathbb{D}$ such that $1a = a$.
8. *distributive*: $a(b + c) = ab + ac$

SOLUTION

In what follows we write $a = a_r + a_d\epsilon$ and likewise for $b, c \in \mathbb{D}$.

Additive associativity and commutativity and existence of additive inverse are both immediate results of dual number addition reducing to element-wise real number addition. Furthermore, by definition of addition on $\mathbb{D}$ the dual number $0 + 0\epsilon$ acts as the additive identity since

$$(a_r + a_d\epsilon) + (0 + 0\epsilon) = (a_r + a_d\epsilon).$$

We explicitly prove multiplicative commutativity

$$ab = (a_r + a_d\epsilon)(b_r + b_d\epsilon) = a_r b_r + (a_r b_d + a_d b_r)\epsilon = b_r a_r + (b_r a_d + b_d a_r)\epsilon = ba.$$

We also explicitly prove multiplicative associativity:

$$(ab)c = ((a_r b_r + (a_r b_d + a_d b_r)\epsilon)c = a_r b_r c_r + ((a_r b_d + a_d b_r)c_r + a_r b_r c_d)\epsilon = a_r b_r c_r + (a_r b_d c_r + a_d b_r c_r + a_r b_r$$

and

$$a(bc) = a((b_r c_r + (b_r c_d + b_d c_r)\epsilon) = a_r b_r c_r + (a_r b_d c_r + a_d b_r c_r + a_r b_r c_d)\epsilon.$$

The number $1 + 0\epsilon$ serves as the multiplicative identity. Note that for any dual number a, we have

$$(1 + 0\epsilon)(a_r + a_d\epsilon) = 1a_r + (a_r 0 + 1a_d)\epsilon = a_r + a_d\epsilon = a.$$

Finally we show distributivity of multiplication:

$$a(b+c) = a(b_r + c_r + (b_d + c_d)\epsilon) = (a_r b_r + a_r c_r) + (a_r b_d + a_r c_d + a_d b_r + a_d c_r)\epsilon,$$

$$ab + ac = a_r b_r + (a_d b_r + a_r b_d)\epsilon + a_r c_r + (a_d c_r + a_r c_d)\epsilon = (a_r b_r + a_r c_r) + (a_r b_d + a_r c_d + a_d b_r + a_d c_r)\epsilon.$$

Problem 2.2⋆ (C) A *field* is a commutative ring such that $0 \neq 1$ and all nonzero elements have a multiplicative inverse, i.e.,
there exists a^{-1} such that $aa^{-1} = 1$. Why isn't $\mathbb{D}$ a field?

SOLUTION

Fields require that all nonzero elements have a unique multiplicative inverse. However, this is not the case for dual numbers. To give an explicit counter example, we show that there is no dual number z which is the inverse of $0 + \epsilon$, i.e. a dual number z such that

$$\frac{(0+\epsilon)}{(z_r + z_d\epsilon)} = 1 + 0\epsilon.$$

By appropriate multiplication with the conjugate we show that

$$\frac{(0+\epsilon)(z_r - z_d\epsilon)}{(z_r + z_d\epsilon)(z_r - z_d\epsilon)} = \frac{z_r\epsilon}{z_r^2} = \frac{\epsilon}{z_r}.$$

This proves that no choice of real part z_r can reach the multiplicative identity $1 + 0\epsilon$ when starting from the number $0 + \epsilon$. More general results for zero real part dual numbers can also be proved.

Problem 2.3⋆ (B) A *matrix representation* of a ring are maps from a group/ring to matrices such that matrix addition and multiplication
behave exactly like addition and multiplication of the ring.
That is, if A and B are elements of the ring and ρ is a representation, then

$$\rho(A + B) = \rho(A) + \rho(B) \text{ and } \rho(AB) = \rho(A)\rho(B).$$

Show that the following are matrix representations of complex numbers and dual numbers (respectively):
$\rho : a + b\mathrm{i} \mapsto \begin{pmatrix} a & b \\ -b & a \end{pmatrix}$,

$\rho_\epsilon : a + b\epsilon \mapsto \begin{pmatrix} a & b \\ 0 & a \end{pmatrix}$.

SOLUTION

Let $A = a + bi$ and $B = c + di$ and ρ be the map to their proposed matrix representations. First we check that addition complies with addition of complex numbers, that is
$(a + bi) + (c + di) = (a + c) + (b + d)i$:

$$\rho(A) + \rho(B) = \begin{pmatrix} a & b \\ -b & a \end{pmatrix} + \begin{pmatrix} c & d \\ -d & c \end{pmatrix} = \begin{pmatrix} a+c & b+d \\ -(b+d) & a+c \end{pmatrix} = \rho(A+B).$$

Next for multiplication we need $(a + bi)(c + di) = (ac - bd) + (bc + ad)i$:

$$\rho(A)\rho(B) = \begin{pmatrix} a & b \\ -b & a \end{pmatrix}\begin{pmatrix} c & d \\ -d & c \end{pmatrix} = \begin{pmatrix} ac - bd & bc + ad \\ -(bc + ad) & ac - bd \end{pmatrix} = \rho(AB)$$

Now we move on to dual numbers $A = a + b\epsilon$ and $B = c + d\epsilon$ and their matrix representation map ρ_ϵ: For addition of dual numbers $(a + b\epsilon) + (c + d\epsilon) = (a + c) + (b + d)\epsilon$:

$$\rho_\epsilon(A) + \rho_\epsilon(B) = \begin{pmatrix} a & b \\ 0 & a \end{pmatrix} + \begin{pmatrix} c & d \\ 0 & c \end{pmatrix} = \begin{pmatrix} a+c & b+d \\ 0 & a+c \end{pmatrix} = \rho_\epsilon(A+B).$$

And finally for multiplication of dual numbers $(a+b\epsilon)(c+d\epsilon) = ac + (ad+bc)\epsilon$:

$$\rho_\epsilon(A)\rho_\epsilon(B) = \begin{pmatrix} a & b \\ 0 & a \end{pmatrix} \begin{pmatrix} c & d \\ 0 & c \end{pmatrix} = \begin{pmatrix} ac & ad+bc \\ 0 & ac \end{pmatrix} = \rho_\epsilon(AB).$$

Problem 2.4⋆ (C) What is the correct definition of division on dual numbers, i.e.,

$$(a+b\epsilon)/(c+d\epsilon) = s + t\epsilon$$

for what choice of s and t? Use dual numbers to compute the derivative of the following functions at $x = 0.1$:

$$\exp(\exp x \cos x + \sin x), \prod_{k=1}^{3}\left(\frac{x}{k} - 1\right), \text{ and } f_2^{\mathrm{s}}(x) = 1 + \frac{x-1}{2 + \frac{x-1}{2}}$$

SOLUTION

As with complex numbers, division is easiest to understand by first multiplying with the conjugate, that is: $\frac{a+b\epsilon}{c+d\epsilon} = \frac{(a+b\epsilon)(c-d\epsilon)}{(c+d\epsilon)(c-d\epsilon)}$.
Expanding the products and dropping terms with ϵ^2 then leaves us with the definition of division for dual numbers (where the denominator must have non-zero real part):
$\frac{a}{c} + \frac{bc-ad}{c^2}\epsilon$. Thus we have $s = \frac{a}{c}$ and $t = \frac{bc-ad}{c^2}$.

We saw in the lectures that we have

$$\exp(a+b\epsilon) = \exp(a) + b\exp(a)\epsilon,$$

$$\sin(a+b\epsilon) = \sin(a) + b\cos(a)\epsilon,$$

and

$$\cos(a+b\epsilon) = \cos(a) - b\sin(a)\epsilon.$$

Using these, we find that evaluating $\exp(\exp x \cos x + \sin x)$ at $a + b\epsilon$ yields

$$\exp((\exp(a) + b\exp(a)\epsilon)(\cos(a) - b\sin(a)\epsilon) + \sin(a) + b\cos(a)\epsilon)$$

$$= \exp(\exp(a)\cos(a) + (\cos(a)\exp(a) - \sin(a)\exp(a))b\epsilon + \sin(a) + b\cos(a)\epsilon) = \exp((\exp(a)\cos(a) + \mathrm{s}$$

which means $\exp(\exp x \cos x + \sin x)$ evaluated at $a + b\epsilon$ is

$$\exp((\exp(a)\cos(a) + \sin(a))) + b(\cos(a)\exp(a) - \sin(a)\exp(a) + \cos(a))\exp((\exp(a)\cos(a) + \sin(a))$$

Setting $b = 1$ the derivative at point a is thus given by the dual part, i.e.:

$$(\cos(a)\exp(a) - \sin(a)\exp(a) + \cos(a))\exp((\exp(a)\cos(a) + \sin(a))).$$

Problem 2.5 (C) Add support for `cos`, `sin`, and `/` to the type `Dual`:

```
# Dual(a,b) represents a + b*ϵ
struct Dual{T}
   a::T
   b::T
end
```

```
# Dual(a) represents a + 0*ϵ
Dual(a::Real) = Dual(a, zero(a)) # for real numbers we use a + 0ϵ

# Allow for a + b*ϵ syntax
const ϵ = Dual(0, 1)

import Base: +, *, -, /, ^, zero, exp, cos, sin, one

# support polynomials like 1 + x, x - 1, 2x or x*2 by reducing to Dual
+(x::Real, y::Dual) = Dual(x) + y
+(x::Dual, y::Real) = x + Dual(y)
-(x::Real, y::Dual) = Dual(x) - y
-(x::Dual, y::Real) = x - Dual(y)
*(x::Real, y::Dual) = Dual(x) * y
*(x::Dual, y::Real) = x * Dual(y)

# support x/2 (but not yet division of duals)
/(x::Dual, k::Real) = Dual(x.a/k, x.b/k)

# a simple recursive function to support x^2, x^3, etc.
function ^(x::Dual, k::Integer)
   if k < 0
       error("Not implemented")
   elseif k == 1
       x
   else
       x^(k-1) * x
   end
end

# support identity of type Dual
one(x::Dual) = Dual(one(eltype(x.a)))

# Algebraic operations for duals
-(x::Dual) = Dual(-x.a, -x.b)
+(x::Dual, y::Dual) = Dual(x.a + y.a, x.b + y.b)
-(x::Dual, y::Dual) = Dual(x.a - y.a, x.b - y.b)
*(x::Dual, y::Dual) = Dual(x.a*y.a, x.a*y.b + x.b*y.a)

exp(x::Dual) = Dual(exp(x.a), exp(x.a) * x.b)

function cos(x::Dual)
   # TODO: implement cos for Duals
   ## SOLUTION
   Dual(cos(x.a), -sin(x.a) * x.b)
   ## END
end

function sin(x::Dual)
   # TODO: implement sin for Duals
   ## SOLUTION
   Dual(sin(x.a), cos(x.a) * x.b)
   ## END
end

function /(x::Dual, y::Dual)
```

```
    # TODO: implement division for Duals
    ## SOLUTION
    if iszero(y.a)
        error("Division for dual numbers is ill-defined when denonimator real part is zero."
    end
    return Dual(x.a / y.a, (y.a * x.b - x.a * y.b) / y.a^2)
    ## END
end
```

Problem 2.6 (B) Use dual numbers to compute the derivatives to

$$\exp\left(\exp x \cos x + \sin x\right), \prod_{k=1}^{1000}\left(\frac{x}{k} - 1\right), \text{ and } f_{1000}^{\mathrm{s}}(x).$$

Does your answer match (to 5 digits) Problem 1.4?

SOLUTION

With the previous problems solved, this is as simple as running

```
fdual = f(0.1+ϵ)
fdual.b
```

```
gdual = g(0.1+ϵ)
gdual.b
```

```
contdual = cont(1000,0.1+ϵ)
contdual.b
```

3. Newton iteration

Newton iteration is an algorithm for computed roots of a function f using its derivative: given an initial guess x_0, one
obtains a sequence of guesses via

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

Problem 3.1 (B) Use `Dual` to implement the following function which returns x_n:

```
function newton(f, x0, n)
    ## TODO: compute x_n
    ## SOLUTION
    xk = x0
    for k = 1:n
        fd = f(xk + ϵ)
        xk = xk - fd.a / fd.b
    end
    return xk
    ## END
end
```

SOLUTION

Here is a simple test case for the above function:

```
# example
f_ex(x) = x^2-3
n = 5
x0 = 1
# compute proposed root
xn = newton(f_ex,x0,n)
# check that obtained point is an approximate root
f_ex(xn)
```

END

Problem 3.2 (B) Compute points y such that $|f(y)| \leq 10^{-13}$ (i.e., approximate roots):

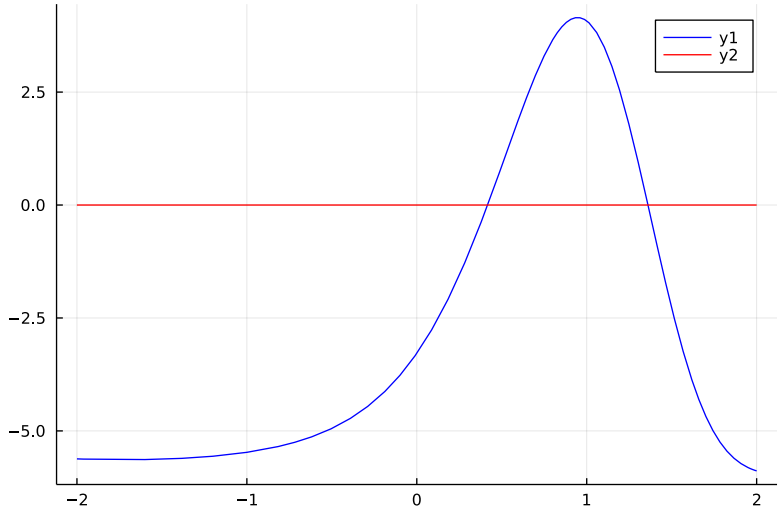
$$\exp\left(\exp x \cos x + \sin x\right) - 6 \text{ and } \prod_{k=1}^{1000} \left(\frac{x}{k} - 1\right) - \frac{1}{2}$$

(Hint: you may need to try different **`x0`** and **`n`** to get convergence. Plotting the function should give an indication of a good initial guess.)

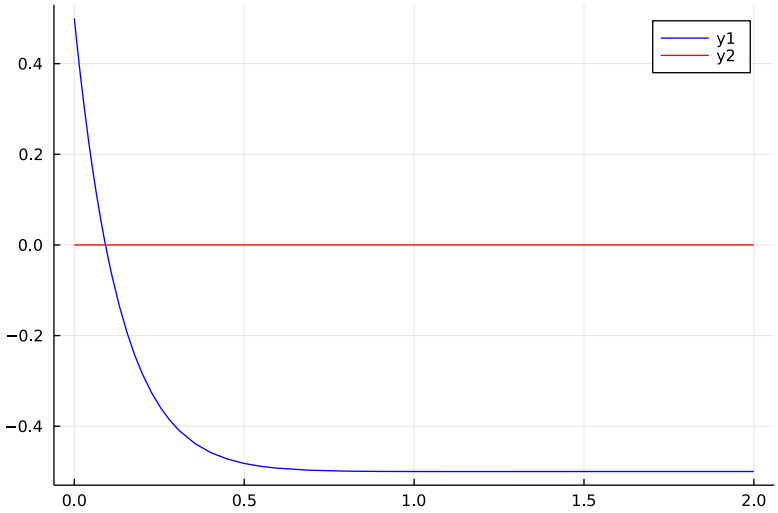
SOLUTION

We can plot the functions on a few ranges to get an intuition

```
using Plots

f1(x) = exp(exp(x)*cos(x) + sin(x)) - 6
f2(x) = prod([x] ./ range(1,1000) .- 1) - 1/2

plot(f1,-2,2,color="blue")
plot!(x->0,color="red")
```




```
plot(f2,0,2,color="blue")
plot!(x->0,color="red")
```



And then use our Newton iteration to compute approximate roots

```
rootf1 = newton(f1, 1.5, 5)
f1(rootf1)
```

```
rootf2 = newton(f2, 0.3, 8)
f2(rootf2)
```

Problem 3.3 (A) Compute points y such that $|f_{1000}^{\mathrm{s}}(y) - j| \leq 10^{-13}$ for $j = 1, 2, 3$.
Make a conjecture of what $f_n^{\mathrm{s}}(x)$ converges to as $n \to \infty$. (Bonus problem: Prove your conjecture.)

SOLUTION

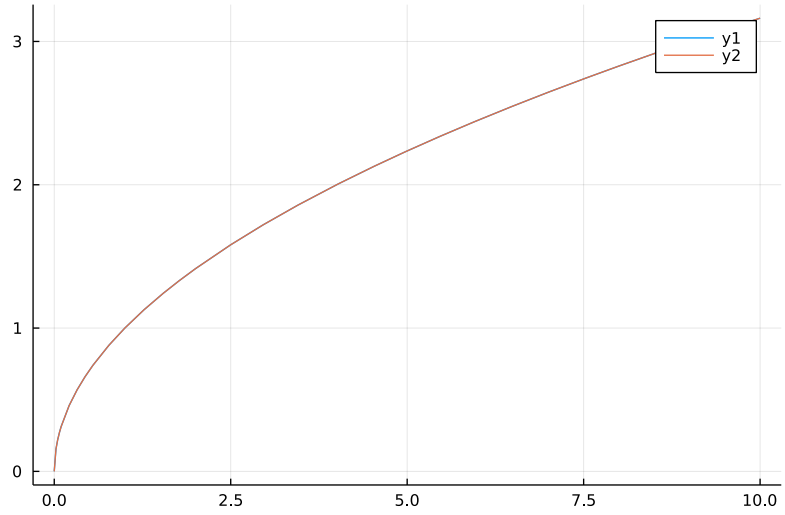
```
xn = newton(x->cont(1000,x)-1.,0.5,10)
cont(1000,xn)-1.
```

```
xn = newton(x->cont(1000,x)-2.,0.5,10)
cont(1000,xn)-2.
```

```
xn = newton(x->cont(1000,x)-3.,0.5,10)
cont(1000,xn)-3.
```

By plotting the function we can conjecture that the continued fraction converges to $\sqrt{x}$:

```
using Plots
plot(x->cont(1000,x),0,10)
plot!(x->sqrt(x))
```



There are a handful of ways to prove this conjecture. Here is one - start with

$$\sqrt{x}(1+\sqrt{x}) = \sqrt{x} + x,$$

then extend the RHS by $0 = +1 - 1$ to also obtain the factor $1+\sqrt{x}$ there, resulting in

$$\sqrt{x}(1+\sqrt{x}) = (1+\sqrt{x}) + x - 1.$$

Dividing through $(1+\sqrt{x})$ now yields

$$\sqrt{x} = 1 + \tfrac{x-1}{1+\sqrt{x}}.$$

Note that we can now plug this equation into itself on the right hand side to obtain a recursive continued fraction for the square root function.

MATH50003 Numerical Analysis: Problem 3

This problem sheet explores implementation of triangular solves, supporting a matrix with two super-diagonals, as well as permutation and Householder reflections that can be applied to a vector in $O(n)$ complexity.

Questions marked with a ⋆ are meant to be completed without using a computer.
Problems are denoted A/B/C to indicate their difficulty.

```
using LinearAlgebra, Test

# We will override these functions below
import Base: getindex, setindex!, size, *, \
```

1. Dense Matrices

Problem 1.1 (C) Show that `A*x` is not implemented as `mul(A, x)` from the lecture notes by finding a `Float64` example where the bits do not match.

SOLUTION

First we have to define `mul(A, x)` as in the lecture notes:

```
function mul(A, x)
   m,n = size(A)
   c = zeros(eltype(x), m) # eltype is the type of the elements of a vector/matrix
   for j = 1:n, k = 1:m
       c[k] += A[k, j] * x[j]
   end
   c
end
```

Then we can easily find examples, in fact we can write a function that searches for examples:

```
using ColorBitstring

function findblasmuldifference(n,l)
for j = 1:n
A = randn(l,l)
x = rand(l)
if A*x != mul(A,x)
return (A,x)
end
end
end

n = 100 # number of attempts
l = 10 # size of objects
(A,x) = findblasmuldifference(n,l) # find a difference

println("Bits of obtained A*x")
printlnbits.(A*x);
println("Bits of obtained mul(A,x)")
printlnbits.(mul(A,x));
println("Difference vector between the two solutions:")
```

```
println(A*x-mul(A,x))
```

2. Triangular Matrices

Problem 2.1 (B) Complete the following functions for solving linear systems with triangular systems by implementing back and forward-substitution:

```
function ldiv(U::UpperTriangular, b)
    n = size(U,1)

    if length(b) != n
        error("The system is not compatible")
    end

    x = zeros(n)  # the solution vector
    ## TODO: populate x using back-substitution
end

function ldiv(U::LowerTriangular, b)
    n = size(U,1)

    if length(b) != n
        error("The system is not compatible")
    end

    x = zeros(n)  # the solution vector
    ## TODO: populate x using forward-substitution
end
```

SOLUTION

```
function ldiv(U::UpperTriangular, b)
    n = size(U,1)

    if length(b) != n
        error("The system is not compatible")
    end

    x = zeros(n)  # the solution vector

    for k = n:-1:1  # start with k=n, then k=n-1, ...
        r = b[k]  # dummy variable
        for j = k+1:n
            r -= U[k,j]*x[j] # equivalent to r = r-U[k,j]*x[j]
        end
        x[k] = r/U[k,k]
    end
    x
end
```

```
function ldiv(U::LowerTriangular, b)
    n = size(U,1)

    if length(b) != n
        error("The system is not compatible")
    end

    x = zeros(n)  # the solution vector

    for k = 1:n  # start with k=1
        r = b[k]  # dummy variable
        for j = 1:k-1
            r -= U[k,j]*x[j]
        end
        x[k] = r/U[k,k]
    end
    x
end
```

Here is an example:

```
x = [1,2,3,4]
Ldense = [1 0 0 0; 2 3 0 0; 4 5 6 0; 7 8 9 10]
Ltriang = LowerTriangular(Ldense)
```

```
Ldense\x-ldiv(Ltriang,x)
```

Problem 2.2⋆ (B) Given $\mathbf{x} \in \mathbb{R}^n$, find a lower triangular matrix of the form

$$L = I - 2\mathbf{v}\mathbf{e}_1^\top$$

such that:

$$L\mathbf{x} = x_1\mathbf{e}_1.$$

What does $L\mathbf{y}$ equal if $\mathbf{y} \in \mathbb{R}^n$ satisfies $y_1 = \mathbf{e}_1^\top \mathbf{y} = 0$?

SOLUTION

By straightforward computation we find

$$Lx = x - 2\mathbf{v}\mathbf{e}_1^\top x = x - 2\mathbf{v}x_1$$

and thus we find such a lower triangular L by choosing $v_1 = 0$ and $v_k = \frac{x_k}{2x_1}$ for $k = 2..n$ and $x_1 \neq 0$.

3. Banded matrices

Problem 3.1 (C) Complete the implementation of **`UpperTridiagonal`** which represents a banded matrix with
bandwidths $(l, u) = (0, 2)$:

```
struct UpperTridiagonal{T} <: AbstractMatrix{T}
   d::Vector{T}   # diagonal entries
   du::Vector{T}  # super-diagonal enries
   du2::Vector{T} # second-super-diagonal entries
end

size(U::UpperTridiagonal) = (length(U.d),length(U.d))

function getindex(U::UpperTridiagonal, k::Int, j::Int)
   d,du,du2 = U.d,U.du,U.du2
   # TODO: return U[k,j]
end

function setindex!(U::UpperTridiagonal, v, k::Int, j::Int)
   d,du,du2 = U.d,U.du,U.du2
   if j > k+2
       error("Cannot modify off-band")
   end

   # TODO: modify d,du,du2 so that U[k,j] == v

   U # by convention we return the matrix
end
```

SOLUTION

```
struct UpperTridiagonal{T} <: AbstractMatrix{T}
   d::Vector{T}   # diagonal entries
   du::Vector{T}  # super-diagonal enries
   du2::Vector{T} # second-super-diagonal entries
end

size(U::UpperTridiagonal) = (length(U.d),length(U.d))

function getindex(U::UpperTridiagonal, k::Int, j::Int)
   d,du,du2 = U.d,U.du,U.du2

   if j == k+2
  return U.du2[k]
   elseif j == k+1
  return U.du[k]
   elseif j == k
  return U.d[k]
   else # off band entries are zero
  return zero(eltype(U))
   end
end

function setindex!(U::UpperTridiagonal, v, k::Int, j::Int)
   d,du,du2 = U.d,U.du,U.du2
   if (j > k+2)||(j<k)
       error("Cannot modify off-band")
   end
```

```
    if j == k+2
   U.du2[k] = v
    elseif j == k+1
   U.du[k] = v
    elseif j == k
   U.d[k] = v
    end

    U # by convention we return the matrix
end
```

We can check that the above methods to read and write entries work:

```
A = UpperTridiagonal([1,2,3,4], [1,2,3], [1,2])
```

```
A[1,1] = 2
A
```

Problem 3.2 (B) Complete the following implementations of `*` and `\` for **`UpperTridiagonal`** so that they take only $O(n)$ operations.

```
function *(U::UpperTridiagonal, x::AbstractVector)
    T = promote_type(eltype(U), eltype(x)) # make a type that contains both the element type
    b = zeros(T, size(U,1)) # returned vector
    # TODO: populate b so that U*x == b (up to rounding)
end

function \(U::UpperTridiagonal, b::AbstractVector)
    T = promote_type(eltype(U), eltype(b)) # make a type that contains both the element type
    x = zeros(T, size(U,2)) # returned vector
    # TODO: populate x so that U*x == b (up to rounding)
end
```

SOLUTION

```
function *(U::UpperTridiagonal, x::AbstractVector)
    T = promote_type(eltype(U), eltype(x)) # make a type that contains both the element type
    b = zeros(T, size(U,1)) # returned vector
    n = size(U)[1]
    for k = 1:n-2
   b[k] = dot(U[k,k:k+2],x[k:k+2])
    end
    # the last two rows need a bit more care
    b[n-1] = dot(U[n-1,n-1:n],x[n-1:n])
    b[n] = U[n,n]*x[n]
    return b
end
```

```
function \(U::UpperTridiagonal, b::AbstractVector)
    T = promote_type(eltype(U), eltype(b)) # make a type that contains both the element type
    x = zeros(T, size(U,2)) # returned vector
    n = size(U)[1]

    if length(b) != n
        error("The system is not compatible")
    end

    for k = n:-1:1  # start with k=n, then k=n-1, ...
        r = b[k]  # dummy variable
        for j = k+1:min(k+3,n)
            r -= U[k,j]*x[j]
        end
        x[k] = r/U[k,k]
    end
    x
end
```

And here is an example of what we have implemented in action:

```
Abanded = UpperTridiagonal([1.1,2.2,3.3,4.4], [1.9,2.8,3.7], [1.5,2.4])
Adense = Matrix(Abanded) # one of many easy ways to convert to dense storage

Adense == Abanded
```

```
x = [5.2,3/4,2/3,9.1415]
Adense*x
```

```
Abanded*x
```

```
Adense\x
```

```
Abanded\x
```

And just for fun, let's do a larger scale dense speed comparison

```
using BenchmarkTools
n = 10000
Abanded = UpperTridiagonal(rand(n),rand(n-1),rand(n-2))
Adense = Matrix(Abanded) # one of many easy ways to convert to dense storage
x = rand(n)

@btime Adense*x;
```

```
@btime Abanded*x;
```

```
@btime Adense\x;
```

```
@btime Abanded\x;
```

4. Permutations

Problem 4.1⋆ (C) What are the permutation matrices corresponding to the following permutations?

$$\begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 1 & 4 & 3 & 6 & 5 \end{pmatrix}.$$

SOLUTION

Let $\sigma = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{pmatrix}$ and $\tau = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 1 & 4 & 3 & 6 & 5 \end{pmatrix}$. There are two ways to construct P_σ and P_τ.

- Column by column:
  $P_\sigma = \begin{pmatrix} \mathbf{e}_{\sigma_1^{-1}} \Big| \mathbf{e}_{\sigma_2^{-1}} \Big| \mathbf{e}_{\sigma_3^{-1}} \end{pmatrix} = (\mathbf{e}_3 | \mathbf{e}_2 | \mathbf{e}_1)$
  $P_\tau = \begin{pmatrix} \mathbf{e}_{\tau_1^{-1}} \Big| \mathbf{e}_{\tau_2^{-1}} \Big| \mathbf{e}_{\tau_3^{-1}} \Big| \mathbf{e}_{\tau_4^{-1}} \Big| \mathbf{e}_{\tau_5^{-1}} \Big| \mathbf{e}_{\tau_6^{-1}} \end{pmatrix} = (\mathbf{e}_2 | \mathbf{e}_1 | \mathbf{e}_4 | \mathbf{e}_3 | \mathbf{e}_6 | \mathbf{e}_5)$
- Row by row:
  $$P_\sigma = I[\sigma, :] = \begin{pmatrix} I[\sigma_1, :] \\ I[\sigma_2, :] \\ I[\sigma_3, :] \end{pmatrix} = \begin{pmatrix} I[3, :] \\ I[2, :] \\ I[1, :] \end{pmatrix}$$
  $$P_\tau = I[\tau, :] = \begin{pmatrix} I[\tau_1, :] \\ I[\tau_2, :] \\ I[\tau_3, :] \\ I[\tau_4, :] \\ I[\tau_5, :] \\ I[\tau_6, :] \end{pmatrix} = \begin{pmatrix} I[2, :] \\ I[1, :] \\ I[4, :] \\ I[3, :] \\ I[6, :] \\ I[5, :] \end{pmatrix}$$

Either way, we have

$$P_\sigma = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \quad \text{and} \quad P_\tau = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Problem 4.2 (B) Complete the implementation of a type representing permutation matrices that supports `P[k,j]` and such that `*` takes $O(n)$ operations.

```
using LinearAlgebra, Test

struct PermutationMatrix <: AbstractMatrix{Int}
```

```
    p::Vector{Int} # represents the permutation whose action is v[p]
    function PermutationMatrix(p::Vector)
        sort(p) == 1:length(p) || error("input is not a valid permutation")
        new(p)
    end
end

function size(P::PermutationMatrix)
  (length(P.p),length(P.p))
end
function getindex(P::PermutationMatrix, k::Int, j::Int)
    # P.p[k] == j ? 1 : 0
    if P.p[k] == j
        1
    else
        0
    end
end
function *(P::PermutationMatrix, x::AbstractVector)
    x[P.p]
end

# If your code is correct, this "unit test" will succeed
p = [1, 4, 2, 5, 3]
P = PermutationMatrix(p)
@test P == I(5)[p,:]
```

The following codes show that `*` takes $O(n)$ of both time and space.

```
using BenchmarkTools, Random
x=0 # for some reason, an error "UndefVarError: x not defined" will occur without this line
for n = 10 .^ (1:7)
    print("n = ", n)
    p = randperm(n)
    P = PermutationMatrix(p)
    x = randn(n)
    @btime P*x
end
```

5. Orthogonal matrices

Problem 5.1⋆ (C) Show that orthogonal matrices preserve the 2-norm of vectors:

$$\|Q\mathbf{v}\| = \|\mathbf{v}\|.$$

SOLUTION

$$\|Q\mathbf{v}\|^2 = (Q\mathbf{v})^\top Q\mathbf{v} = \mathbf{v}^\top Q^\top Q\mathbf{v} = \mathbf{v}^\top\mathbf{v} = \|\mathbf{v}\|^2$$

∎

Problem 5.2⋆ (B) Show that the eigenvalues λ of an orthogonal matrix Q are on the unit circle: $|\lambda| = 1$.

SOLUTION

Let $\mathbf{v}$ be a unit eigenvector corresponding to λ: $Q\mathbf{v} = \lambda\mathbf{v}$ with $\|\mathbf{v}\| = 1$. Then

$$1 = \|\mathbf{v}\| = \|Q\mathbf{v}\| = \|\lambda\mathbf{v}\| = |\lambda|.$$

∎

Problem 5.3⋆ (A) Explain why an orthogonal matrix Q must be equal to I if all its eigenvalues are 1.

SOLUTION

Note that Q is normal ($Q^\top Q = I$) and therefore by the spectral theorem for normal matrices we have

$$Q = Q\Lambda Q^\star = QQ^\star = I$$

since Q is unitary.

Problem 5.4 (B) Complete the implementation of a type representing reflections that supports `Q[k,j]` and such that `*` takes $O(n)$ operations.

```
using LinearAlgebra, Test

# Represents I - 2v*v'
struct Reflection{T} <: AbstractMatrix{T}
   v::Vector{T}
end

Reflection(x::Vector{T}) where T = Reflection{T}(x/norm(x))

function size(Q::Reflection)
  (length(Q.v),length(Q.v))
end
function getindex(Q::Reflection, k::Int, j::Int)
  (k == j) - 2Q.v[k]*Q.v[j] # note true is treated like 1 and false like 0
end
function *(Q::Reflection, x::AbstractVector)
   x - 2*Q.v * dot(Q.v,x) # (Q.v'*x) also works instead of dot
end

# If your code is correct, these "unit tests" will succeed
x = randn(5)
Q = Reflection(x)
v = x/norm(x)
@test Q == I-2v*v'
@test Q*v ≈ -v
@test Q'Q ≈ I
```

The following codes show that `*` takes $O(n)$ of both time and space.

```
using BenchmarkTools, Random
for n = 10 .^ (1:7)
   print("n = ", n)
   x = randn(n)
   Q = Reflection(x)
   v = randn(n)
   @btime Q*v
end
```

Problem 5.5 (C) Complete the following implementation of a Housholder reflection so that the unit tests pass. Here `` `s == true` `` means the Householder reflection is sent to the positive axis and `` `s == false` `` is the negative axis.

```
function householderreflection(s::Bool, x::AbstractVector)
   y = copy(x) # don't modify `x`
   if s
       y[1] -= norm(x)
   else
       y[1] += norm(x)
   end
   Reflection(y)
end

x = randn(5)
Q = householderreflection(true, x)
@test Q isa Reflection
@test Q*x ≈ [norm(x);zeros(eltype(x),length(x)-1)]

Q = householderreflection(false, x)
@test Q isa Reflection
@test Q*x ≈ [-norm(x);zeros(eltype(x),length(x)-1)]
```

Problem 5.6⋆ (A) Consider a Householder reflection with $\mathbf{x} = [1, h]$ with $h = 2^{-n}$. What is the floating point error in computing $\mathbf{y} = \mp\|\mathbf{x}\|\mathbf{e}_1 + \mathbf{x}$ for each choice of sign.

SOLUTION

Since $\|\mathbf{x}\| = \sqrt{1+h^2}$, we have $\mathbf{y} = [1 \mp \sqrt{1+h^2}, h]$. We note first that h^{fp} and $(h^2)^{fp}$ are exact due to the choice of h, so we only need to discuss the floating error in computing $1 \mp \sqrt{1+h^2}$.

Numerically, let the length of the significand be S, then

$$1 \oplus h^2 = \begin{cases} 1+h^2 & n \le S/2 \\ 1 & n > S/2 \end{cases} = 1 + h^2 + \delta_1$$

where $|\delta_1| \le \frac{\epsilon_m}{2}$.

$+$ PLUS $+$

Since $\sqrt{1 \oplus h^2}^{fp} > 0$, we know that

$$\begin{aligned}1 \oplus \sqrt{1 \oplus h^2}^{fp} =& (1+\delta_2)(1+\sqrt{1 \oplus h^2}^{fp}) \\ =& (1+\delta_2)(1+\sqrt{1+h^2+\delta_1}(1+\delta_3))\end{aligned}$$

where $|\delta_2|, |\delta_3| \le \frac{\epsilon_m}{2}$. Then

$$\begin{aligned}\frac{1 \oplus \sqrt{1 \oplus h^2}^{fp}}{1+\sqrt{1+h^2}} =& (1+\delta_2)\left(1 + \frac{\sqrt{1+h^2+\delta_1}(1+\delta_3) - \sqrt{1+h^2}}{1+\sqrt{1+h^2}}\right) \\ =& (1+\delta_2)\left(1 + \frac{(1+\delta_3)(\sqrt{1+h^2+\delta_1} - \sqrt{1+h^2}) + \delta_3\sqrt{1+h^2}}{1+\sqrt{1+h^2}}\right) \\ \approx& (1+\delta_2)\left(1 + \frac{\delta_1}{2(1+\sqrt{1+h^2})\sqrt{1+h^2}} + \delta_3\frac{\sqrt{1+h^2}}{1+\sqrt{1+h^2}}\right)\end{aligned}$$

and we can bound the relative error by

$$|\delta_2| + |\delta_1|\tfrac{1}{2(1+\sqrt{1+h^2})\sqrt{1+h^2}} + |\delta_3|\tfrac{\sqrt{1+h^2}}{1+\sqrt{1+h^2}} \le |\delta_2| + \tfrac{|\delta_1|}{4} + \tfrac{3|\delta_3|}{4} \le \epsilon_m.$$

In conclusion, it's very accurate to compute $1+\sqrt{1+h^2}$. Let us verify this:

```
using Plots
S = precision(Float64)-1;
relative_error = zeros(S)
for n = 1:S
   h = 2.0^(-n)
   exact = 1+sqrt(1+BigFloat(h)^2)
   numerical = 1+sqrt(1+h^2)
   relative_error[n] = abs(numerical-exact)/exact
end
println(eps())
maximum(relative_error)
```

– MINUS –

If $n > S/2$, then $1 \ominus \sqrt{1 \oplus h^2}^{fp} = 1 \ominus \sqrt{1}^{fp} = 1 \ominus 1 = 0$ so the relative error is 100%.

If $n \le S/2$ but not too small, $1 \oplus h^2$ is exactly $1+h^2$ but $\sqrt{1+h^2}^{fp}$ can have rounding error. Expand $\sqrt{1+h^2}$ into Taylor series:

$$\sqrt{1+h^2} = 1 + \tfrac{1}{2}h^2 - \tfrac{1}{8}h^4 + \tfrac{1}{16}h^6 - O(h^8) = 1 + 2^{-2n-1} - 2^{-4n-3} + 2^{-6n-4} - O(2^{-8n})$$

so

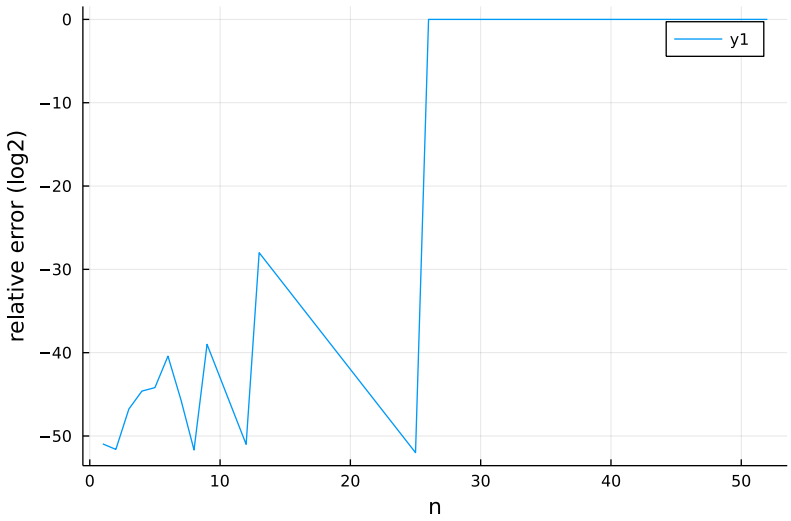
$$\sqrt{1+h^2}^{fp} = \begin{cases} 1 & n = S/2 \\ 1 + \frac{1}{2}h^2 & \frac{S-3}{4} \le n < S/2 \\ 1 + \frac{1}{2}h^2 - \frac{1}{8}h^4 & \frac{S-4}{6} \le n < \frac{S-3}{4} \\ \vdots & \vdots \end{cases}$$

where we can conclude that the absolute error is approximately $\frac{1}{2}h^2, \frac{1}{8}h^4, \frac{1}{16}h^6, \ldots$ for each stage when h is small. Keeping in mind that $1 - \sqrt{1+h^2} \approx -\frac{1}{2}h^2$ when h is small, the relative error is approximately $1, \frac{1}{4}h^2, \frac{1}{8}h^4, \ldots$ for each stage. Special note: the relative error is exactly 1 in the first stage when $n = S/2$.

If n is so small that $\sqrt{1+h^2}$ is noticably larger than 1, the absolute error can be bounded by $\frac{\epsilon_m}{2}$ so the relative error is bounded by $\frac{\epsilon_m}{2(\sqrt{1+h^2}-1)} \approx \frac{\epsilon_m}{h^2}$.

Let us verify these conclusions:

```
using Plots
S=precision(Float64)-1;
relative_error=zeros(S)
for n=1:S
   h=2.0^(-n)
   exact=1-sqrt(1+BigFloat(h)^2)
   numerical=1-sqrt(1+h^2)
   relative_error[n]=-abs(numerical-exact)/exact
end
plot(1:S,log2.(relative_error), xlabel="n", ylabel="relative error (log2)")
```



From this plot we can clearly identify the 3 phases:

1. When n is very small, the relative error grows smoothly with n;
2. When n is neither too large nor too small, the relative error has jumps at around $S/2 = 26$, $(S-3)/4 = 12.25$, $(S-4)/6 = 8$. In each stage, the slope is as predicted. For example, the first stage from $S/2$ to $(S-3)/4$ has relative error of $\frac{1}{4}h^2$, hence the slope is -2 between 13 and 25.
3. When $n \geq S/2$, the relative error is 100%;

The transition point between phase 1 and 2 is at the stage for $O(h^8)$ of relative error from phase 2 which meets $\frac{\epsilon_m}{h^2}$ from phase 1.

MATH50003 Numerical Analysis: Problem Sheet 4

This problem sheet explores least squares, the QR decomposition including for
tridiagonal matrices,
and the PLU decompositions.

Questions marked with a ⋆ are meant to be completed without using a computer.
Problems are denoted A/B/C to indicate their difficulty.

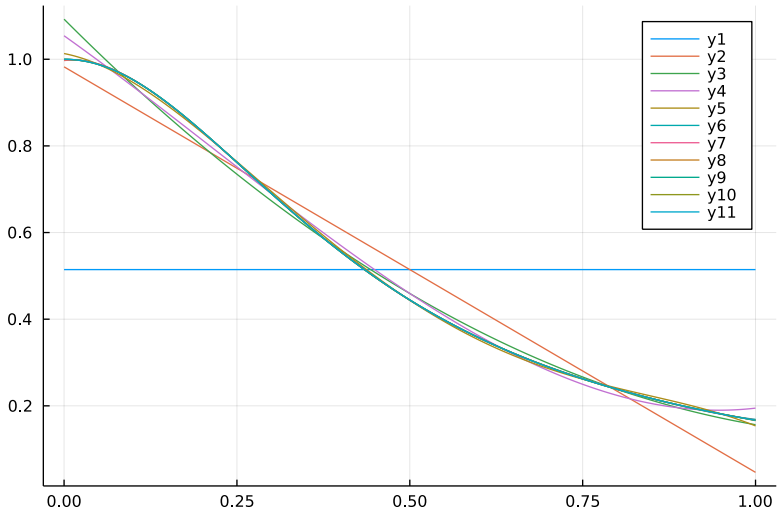
```
using LinearAlgebra, Plots, Test
```

1. Least squares and QR decompositions

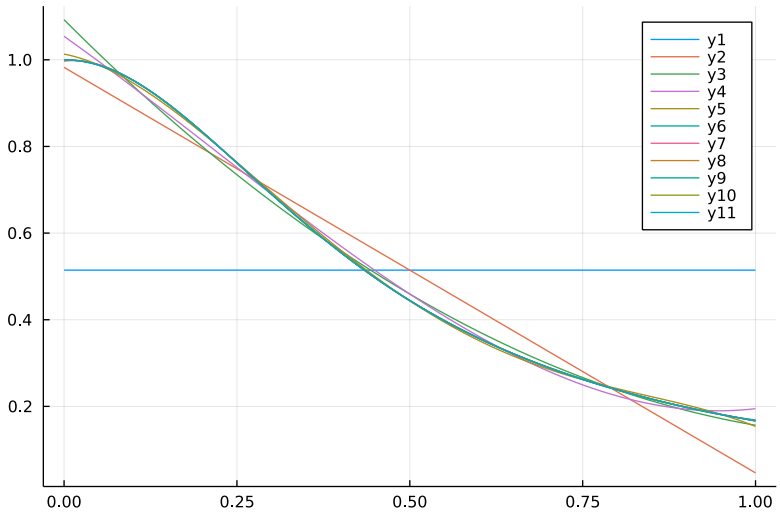
Problem 1.1 (B) Find and plot the best least squares fit of $\frac{1}{5x^2+1}$ by degree n polynomials for $n = 0, \dots, 10$ at 1000 evenly spaced points between 0 and 1.

SOLUTION

```
x = range(0, 1; length=1000)
pl = plot()
f = 1 ./ (5x.^2 .+ 1)
for n = 0:10
   # Creates 1000 x n+1 matrix with entries x[k]^(j-1)
   A = x .^ (0:n)'
   c = A \ f
   plot!(x, A*c)
end
pl
```




```
# Here is the same approach as above but this time explicitly using QR as discussed in the
x = range(0, 1; length=1000)
pl = plot()
f = 1 ./ (5x.^2 .+ 1)
for n = 0:10
   # Creates 1000 x n+1 matrix with entries x[k]^(j-1)
   A = x .^ (0:n)'
  (Q, R) = qr(A)
   c = R \ Q[:, 1:n+1]' * f
   plot!(x, A * c)
end
pl
```



END

Problem 1.2⋆ (B) Show that every matrix has a QR decomposition such that the diagonal of R is non-negative.
Make sure to include the case of more columns than rows.

SOLUTION

Beginning with the square case, a square matrix $A = QR$ with square orthogonal Q and square upper triangular R can always be rewritten in the form $A = QD^{-1}DR$ where D is a diagonal matrix with $sign(R[j,j])$ on the diagonal. As a result, DR is an upper triangular matrix with positive diagonal. It remains to check that $QD^{-1} = -QD$ is still orthogonal - this is easy to check since $-QD(-QD)^T = QDDQ^T = QQ^T = I$.
Note we have made use of the fact that the inverse of a diagonal matrix is diagonal, that any diagonal matrix satisfies $D^T = D$ and that $DD = I$ since $\operatorname{sign}(R[j,j])^2 = 1$.

The same argument works for the non-square cases as long as we take care to consider the appropriate dimensions and pad with the identity matrix. Note that Q is always a square matrix in the QR decomposition. Assume the R factor has more columns m than rows n. Then a square $n \times n$ diagonal matrix with its n diagonal entries being $\operatorname{sign}(R[j,j]), j = 1..n$ works with the same argument as above.

Finally, assume that R has less colums m than rows n. In this case the square $n \times n$ diagonal matrix with its first m diagonal entries being $\operatorname{sign}(R[j,j]), j = 1..m$ and any remaining diagonal entries being 1 works with the same argument as above.

Since the matrix D is always square diagonal and orthogonal by construction, everything else for both of these cases is exactly as in the square case.

Here are some practical examples in Julia:

```
E = [-1 0 0 0; 0 1 0 0; 0 0 1 0; 0 0 0 1]
R = [-1 3; 0 2; 0 0; 0 0]
```

```
E*R
```

```
E = [-1 0 0 0; 0 1 0 0; 0 0 -1 0; 0 0 0 1]
R = [-1 3 7 8 2 1; 0 2 9 2 1 9; 0 0 -7 2 1 9; 0 0 0 7 5 2]
```

```
E*R
```

END

Problem 1.3⋆ (B) Show that the QR decomposition of a square invertible matrix is unique, provided that the diagonal of R is positive.

SOLUTION

Assume there is a second decomposition also with positive diagonal

$$A = QR = \tilde{Q}\tilde{R}$$

Then we know

$$Q^\top \tilde{Q} = R\tilde{R}^{-1}$$

Note $Q^\top \tilde{Q}$ is orthogonal, and $R\tilde{R}^{-1}$ has positive eigenvalues (the diagonal), hence all m eigenvalues of $Q^\top \tilde{Q}$ are 1. This means that $Q^\top \tilde{Q} = I$ and hence $\tilde{Q} = Q$, which then implies $\tilde{R} = R$.∎
END

2. Gram–Schmidt

Problem 2.1⋆ (B) The modified Gram–Schmidt algorithm is a slight variation of Gram–Schmidt where instead of computing

$$\mathbf{v}_j := \mathbf{a}_j - \sum_{k=1}^{j-1} \underbrace{\mathbf{q}_k^\top \mathbf{a}_j}_{r_{kj}} \mathbf{q}_k$$

we compute it step-by-step:

$$\begin{aligned} \mathbf{v}_j^1 &:= \mathbf{a}_j \\ \mathbf{v}_j^{k+1} &:= \mathbf{v}_j^k - \mathbf{q}_k^\top \mathbf{v}_j^k \mathbf{q}_k \end{aligned}$$

Show that $\mathbf{v}_j^j = \mathbf{v}_j$.

SOLUTION

Recall that the column vectors of Q are orthonormal i.e. $(\mathbf{q}_i, \mathbf{q}_j) = \delta_{ij}$.
Next observe that substituting from $\mathbf{v}_j^1$ for $\mathbf{a}_j$, for each $\mathbf{v}_j^i$ we get i terms (note that the second term in the definition of $\mathbf{v}_j^{k+1}$ contributes only 1 term by orthonormality of Q, retrieving the contribution from $\mathbf{v}_j^{i-1}$ plus one.)

$$\mathbf{v}_j^i := \mathbf{a}_j - \sum_{k=1}^{i-2} q_k^\top \mathbf{a}_j q_k - q_{i-1}^\top \mathbf{a}_j q_{i-1}$$

END

Problem 2.2 (B) Complete the following
function implementing the modified Gram–Schmidt algorithm:

```
function modifiedgramschmidt(A)
    m,n = size(A)
    m ≥ n || error("Not supported")
    R = zeros(n,n)
    Q = zeros(m,n)
    for j = 1:n
        # TODO: Implement the Modified Gram-Schmidt algorthm
    end
    Q,R
end

A = randn(300,300)
Q,R = modifiedgramschmidt(A)
@test A ≈ Q*R
@test Q'Q ≈ I
```

SOLUTION

```
function modifiedgramschmidt(A)
    m,n = size(A)
    m ≥ n || error("Not supported")
    R = zeros(n,n)
    Q = zeros(m,n)
    # looping through the columns
    for j = 1:n
        v = A[:,j]
        # now go through each row
        for k = 1:j-1
            # take inner product with q_k and v
            R[k, j] = Q[:, k]' * v
            v = v - R[k, j] * Q[:, k]
        end
        R[j,j] = norm(v)
        Q[:,j] = v/R[j,j]
    end
    Q,R
end

A = randn(300,300)
Q,R = modifiedgramschmidt(A)
@test A ≈ Q*R
@test Q'Q ≈ I
```

END

Problem 2.3 (B) Compare the orthogonality of `Q` between `gramschmidt` and `modifiedgramschmidt` when applied to a `300 × 300` random matrix.

SOLUTION

For reference, here is the gramschmidt algorithm from the lecture notes:

```
function gramschmidt(A)
   m,n = size(A)
   m ≥ n || error("Not supported")
   R = zeros(n,n)
   Q = zeros(m,n)
   for j = 1:n
       for k = 1:j-1
           R[k,j] = Q[:,k]'*A[:,j]
       end
       v = A[:,j] - Q[:,1:j-1]*R[1:j-1,j]
       R[j,j] = norm(v)
       Q[:,j] = v/R[j,j]
   end
   Q,R
end

# Now compare MGS with GS:

A = randn(300,300)
Q,R = gramschmidt(A)
Qm,Rm = modifiedgramschmidt(A)

maximum(abs.(Q*Q')-I)
```

Modified Gram–Schmidt is more accurate for constructing an orthogonal matrix:

```
maximum(abs.(Qm*Qm')-I)
```

END

3. Householder reflections

Problem 3.1 (B)

Complete the definition of `Reflections` which supports a sequence of reflections, that is,

$$Q = Q_{\mathbf{v}_1} \cdots Q_{\mathbf{v}_n}$$

where the vectors are stored as a matrix `V` whose j-th column is $\mathbf{v}_j$, and

$$Q_{\mathbf{v}_j} = I - 2\mathbf{v}_j \mathbf{v}_j^\top.$$

```
struct Reflections{T} <: AbstractMatrix{T}
   V::Matrix{T}
end

import Base: *, size, getindex

size(Q::Reflections) = (size(Q.V,1), size(Q.V,1))
```

```
function *(Q::Reflections, x::AbstractVector)
    T = eltype(Q)
    r = Vector{T}(x) # convert x to a vector of type T
    # TODO: Apply Q in O(mn) operations by applying
    # the reflection corresponding to each column of Q.V to r

    ## SOLUTION
    m,n = size(Q.V)
    for j = n:-1:1
        v = Q.V[:, j]
        r = r - 2 * v * dot(v, r)
    end
    ## END

    r
end

function getindex(Q::Reflections, k::Int, j::Int)
    # TODO: Return Q[k,j] in O(mn) operations (hint: use *)

    ## SOLUTION
    T = eltype(Q.V)
    m,n = size(Q)
    ej = zeros(T, m)
    ej[j] = one(T)
    return (Q*ej)[k]
    ## END
end

Y = randn(5,3)
V = Y * Diagonal([1/norm(Y[:,j]) for j=1:3])
Q = Reflections(V)
@test Q ≈ (I - 2V[:,1]*V[:,1]')*(I - 2V[:,2]*V[:,2]')*(I - 2V[:,3]*V[:,3]')
@test Q'Q ≈ I
```

Problem 3.2 (B) Complete the following function that implements Householder QR using only $O(mn^2)$ operations.

```
function householderqr(A)
    m,n = size(A)
    if m < n
        error("Only support more rows than columns")
    end
    # R begins as A, modify it in place
    R = copy(A)
    Q = Reflections(Matrix(1.0I, m, n))
    for j = 1:n
        # TODO: populate Q and R using O(m*(n-j)) operations
        ## SOLUTION
        y = copy(R[j:end, j])
        y[1] += ((y[1] ≥ 0) ? 1 : -1) * norm(y)
        w = y / norm(y)
```

```
        Q.V[j:end, j] = w
        R[j:end, :] = R[j:end, :] - 2 * w * (w' * R[j:end, :])
        ## END
    end
    Q,R
end

A = randn(4,6)
Q,R = householderqr(A)
@test Q*R ≈ A
@test Q'Q ≈ I
```

SOLUTION

Here we sketch some additional notes comparing performance.

```
# Here's a performance comparison with what we saw in the lecture notes:
function householderreflection(x)
    y = copy(x)
    # we cannot use sign(x[1]) in case x[1] == 0
    if x[1] ≥ 0
        y[1] += norm(x)
    else
        y[1] -= norm(x)
    end
    w = y / norm(y)
    I - 2 * w * w'
end
function lecturehouseholderqr(A)
    m,n = size(A)
    R = copy(A)
    Q = Matrix(1.0I, m, m)
    for j = 1:n
        Qⱼ = householderreflection(R[j:end,j])
        R[j:end,:] = Qⱼ*R[j:end,:]
        Q[:,j:end] = Q[:,j:end]*Qⱼ
    end
    Q,R
end

using BenchmarkTools

for j = 1:5
    A = randn(j*100,j*100)
    @btime lecturehouseholderqr(A);
end
```

```
for j = 1:5
    A = randn(j*100,j*100)
    @btime householderqr(A);
end
```

END

4. Banded QR with Given's rotations

Problem 4.1⋆ (A) Describe an algorithm for computing the QR decomposition of a tridiagonal matrix using rotations instead of reflections to upper-triangularise column-by-column.

SOLUTION

Let A be a tridiagonal matrix:

$$A = \begin{bmatrix} a_{11} & a_{12} & 0 & \cdots & 0 \\ a_{21} & a_{22} & a_{23} & \ddots & \vdots \\ 0 & a_{32} & a_{33} & \ddots & 0 \\ & 0 & \ddots & \ddots & a_{n-1,n} \\ & & 0 & a_{n,n-1} & a_{nn} \end{bmatrix} = \begin{bmatrix} \mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_n \end{bmatrix},$$

where each $\mathbf{a}_j \in \mathbb{R}^n$ and $[\mathbf{a}_j]_k = 0$ for $|j-k| > 1$.

Recall that,

$$\frac{1}{\sqrt{a^2+b^2}} \begin{bmatrix} a & b \\ -b & a \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} \sqrt{a^2+b^2} \\ 0 \end{bmatrix},$$

and that,

$$\frac{1}{\sqrt{a^2+b^2}} \begin{bmatrix} a & b \\ -b & a \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix},$$

is a rotation matrix where $\theta = -\arctan(b/a)$. With this in mind, consider multiplying A from the left by,

$$Q_1 = \begin{bmatrix} \frac{a_{11}}{r_{11}} & \frac{a_{21}}{r_{11}} & & & \\ -\frac{a_{21}}{r_{11}} & \frac{a_{11}}{r_{11}} & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \end{bmatrix},$$

where $r_{11} = \sqrt{a_{11}^2 + a_{21}^2}$. This rotates dimensions 1 and 2 through angle $\theta = -\arctan(a_{21}/a_{11})$. We have,

$$Q_1\mathbf{a}_1 = \begin{bmatrix} r_{11} \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \qquad Q_1\mathbf{a}_2 = \begin{bmatrix} r_{12} := \frac{1}{r_{11}}(a_{11}a_{12} + a_{21}a_{22}) \\ t_1 := \frac{1}{r_{11}}(a_{11}a_{22} - a_{21}a_{12}) \\ a_{32} \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \qquad Q_1\mathbf{a}_3 = \begin{bmatrix} r_{13} := \frac{1}{r_{11}}a_{21}a_{23} \\ s_1 := \frac{1}{r_{11}}a_{11}a_{23} \\ a_{33} \\ a_{43} \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \qquad Q_1\mathbf{a}_k = \mathbf{a}_k$$

Then we take,

$$Q_2 = \begin{bmatrix} 1 & & & & \\ & \frac{t_1}{r_{22}} & \frac{a_{32}}{r_{22}} & & \\ & -\frac{a_{32}}{r_{22}} & \frac{t_1}{r_{22}} & & \\ & & & 1 & \\ & & & & \ddots \\ & & & & & 1 \end{bmatrix},$$

where $r_{22} = \sqrt{t_1^2 + a_{32}^2}$, a matrix which rotates dimensions 2 and 3 through angle $\theta_2 = -\arctan(a_{32}/t_1)$. Then,

$$Q_2Q_1\mathbf{a}_1 = Q_1\mathbf{a}_1 = \begin{bmatrix} r_{11} \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \qquad Q_2Q_1\mathbf{a}_2 = Q_2\begin{bmatrix} r_{12} \\ t_1 \\ a_{32} \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} r_{12} \\ r_{22} \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \qquad Q_2Q_1\mathbf{a}_3 = Q_2\begin{bmatrix} r_{13} \\ s_1 \\ a_{33} \\ a_{43} \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} r_{13} \\ r_{23} := \frac{1}{r_{22}}(t_1s_1 - \\ t_2 := \frac{1}{r_{22}}(t_1a_{33} \\ a_{43} \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

$$Q_2Q_1\mathbf{a}_4 = Q_2\begin{bmatrix} 0 \\ 0 \\ a_{34} \\ a_{44} \\ a_{54} \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ r_{24} := \frac{1}{r_{22}}a_{32}a_{34} \\ s_2 := \frac{1}{r_{22}}t_1a_{34} \\ a_{44} \\ a_{54} \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \qquad Q_2Q_1\mathbf{a}_k = \mathbf{a}_k \text{ for } k > 4.$$

Now, for $j = 3 \to n-1$ we take,

$$Q_j := \begin{bmatrix} \mathbf{I}_{j-1} & & & \\ & \frac{t_{j-1}}{r_{jj}} & \frac{a_{j+1,j}}{r_{jj}} & \\ & \frac{a_{j+1,j}}{r_{jj}} & \frac{t_{j-1}}{r_{jj}} & \\ & & & \mathbf{I}_{n-j-1} \end{bmatrix},$$

where $r_{jj} := \sqrt{t_{j-1}^2 + a_{j+1,j}^2}$.

This gives,

$$Q_j \dots Q_1\mathbf{a}_k = \begin{bmatrix} \mathbf{0}_{k-3} \\ r_{k-2,k} \\ r_{k-1,k} \\ r_{k,k} \\ \mathbf{0}_{n-k} \end{bmatrix},$$

for $k \le j$,
and,

$$Q_j \dots Q_1 \mathbf{a}_{j+1} = \begin{bmatrix} \mathbf{0} \\ r_{j-1,j+1} \\ r_{j,j+1} := \frac{1}{r_{jj}}(t_{j-1}s_{j-1} + a_{j+1,j}a_{j+1,j+1}) \\ t_j := \frac{1}{r_{jj}}(t_{j-1}a_{j+1,j+1} - s_{j-1}a_{j+1,j}) \\ a_{j+2,j+1} \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \qquad Q_j \dots Q_1 \mathbf{a}_{j+2} = \begin{bmatrix} \mathbf{0} \\ r_{j,j+2} := \frac{1}{r_{jj}}a_{j+1,j}a_{j+1,j+2} \\ s_j := \frac{1}{r_{jj}}t_{j-1}a_{j+1,j+2} \\ a_{j+2,j+2} \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

Finally we define, $r_{nn} = \frac{1}{r_{n-1,n-1}}(t_{n-2}a_{n,n} - a_{n,n-1}s_{n-2})$, to obtain,

$$Q_{n-1} \dots Q_1 A = \begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 & \dots & 0 \\ 0 & r_{22} & r_{23} & r_{24} & & 0 \\ & \ddots & \ddots & \ddots & \ddots & \vdots \\ & & 0 & r_{n-2,n-2} & r_{n-2,n-1} & r_{n-2,n} \\ & & & 0 & r_{n-1,n-1} & r_{n-1,n} \\ & & & & 0 & r_{n,n} \end{bmatrix} =: R$$

so that $A = QR$, for $Q = Q_1^{-1} \dots Q_{n-1}^{-1}$, where each matrix Q_j rotates the coordinates $(j, j+1)$ through the angle $\theta_j = -\arctan(a_{j+1,j}/t_{j-1})$, and thus each matrix Q_j^{-1} rotates the coordinates $(j, j+1)$ through the angle $\arctan(a_{j+1,j}/t_{j-1})$. This will be important for Problem 4.3.

END

Problem 4.2 (B) Implement `Rotations` which represents an orthogonal matrix `Q` that is a product of rotations of angle `θ[k]`, each acting on the entries `k:k+1`. That is, it returns $Q = Q_1 \cdots Q_k$ such that

$$Q_k[k:k+1, k:k+1] = [cos(\theta[k]) - sin(\theta[k])sin(\theta[k])cos(\theta[k])]$$

```
struct Rotations{T} <: AbstractMatrix{T}
   θ::Vector{T}
end

import Base: *, size, getindex

size(Q::Rotations) = (length(Q.θ)+1, length(Q.θ)+1)


function *(Q::Rotations, x::AbstractVector)
   T = eltype(Q)
   y = convert(Vector{T}, x)
   # TODO: Apply Q in O(n) operations, modifying y in-place

   ## SOLUTION
   θ = Q.θ
   #Does Q1....Qn x
   for k = length(θ):-1:1
       #below has 4 ops to make the matrix and 12 to do the matrix-vector multiplication,
       #total operations will be 48n = O(n)
       c, s = cos(θ[k]), sin(θ[k])
       y[k:(k+1)] = [c -s; s c] * y[k:(k+1)]
   end
   ## END
```

```
    y
end

function getindex(Q::Rotations, k::Int, j::Int)
    # TODO: Return Q[k,j] in O(n) operations (hint: use *)

    ## SOLUTION
    #recall that A_kj = e_k'*A*e_j for any matrix A
    #so if we use * above, this will take O(n) operations
    n = size(Q)[1]
    ej = zeros(eltype(Q), n)
    ej[j] = 1
    #note, must be careful to ensure that ej is a VECTOR
    #not a MATRIX, otherwise * above will not be used
    Qj = Q * ej
    Qj[k]
    ## END
end

θ1 = randn(5)
Q = Rotations(θ1)
@test Q'Q ≈ I
@test Rotations([π/2, -π/2]) ≈ [0 0 -1; 1 0 0; 0 -1 0]
```

Problem 4.3 (A) Combine **`Rotations`** and **`UpperTridiagonal`** from last problem sheet to implement a banded QR decomposition, **`bandedqr`**, that only takes $O(n)$ operations. Hint: use **`atan(y,x)`** to determine the angle.

```
# First we include UpperTridiagonal from last problem sheet.
# bandedqr is below.
import Base: *, size, getindex, setindex!
struct UpperTridiagonal{T} <: AbstractMatrix{T}
    d::Vector{T}    # diagonal entries
    du::Vector{T}   # super-diagonal enries
    du2::Vector{T} # second-super-diagonal entries
end

size(U::UpperTridiagonal) = (length(U.d),length(U.d))

function getindex(U::UpperTridiagonal, k::Int, j::Int)
    d,du,du2 = U.d,U.du,U.du2
    # TODO: return U[k,j]
    if j - k == 0
        d[j]
    elseif j - k == 1
        du[k]
    elseif j - k == 2
        du2[k]
    else
        0
    end
```

```
end

function setindex!(U::UpperTridiagonal, v, k::Int, j::Int)
    d,du,du2 = U.d,U.du,U.du2
    if j > k+2
        error("Cannot modify off-band")
    end

    # TODO: modify d,du,du2 so that U[k,j] == v
    if j - k == 0
        d[k] = v
    elseif j - k == 1
        du[k] = v
    elseif j - k == 2
        du2[k] = v
    else
        error("Cannot modify off-band")
    end
    U = UpperTridiagonal(d,du,du2)

    U # by convention we return the matrix
end

function bandedqr(A::Tridiagonal)
    n = size(A, 1)
    Q = Rotations(zeros(n - 1)) # Assume Float64
    R = UpperTridiagonal(zeros(n), zeros(n - 1), zeros(n - 2))
    R[1, 1:2] = A[1, 1:2]

    for j = 1:n-1
        # angle of rotation
        Q.θ[j] = atan(A[j+1, j], R[j, j])
        θ = -Q.θ[j] # rotate in oppoiste direction

        c, s = cos(θ), sin(θ)
        # [c -s; s c] represents the rotation that introduces a zero.

        ## TODO: modify rows k and k+1 of R to represent
        # applying the rotation. Note you will need to use row k+1 of A.

        ## SOLUTION
        # This is [c -s; s c] to j-th column, but ignore second row
        # which is zero
        R[j, j] = c * R[j, j] - s * A[j+1, j]
        # This is [c -s; s c] to (j+1)-th column
        R[j:j+1, j+1] = [c -s; s c] * [R[j, j+1]; A[j+1, j+1]]

        if j < n - 1
            # This is [c -s; s c] to (j+2)-th column, where R is still zero
            R[j:j+1, j+2] = [-s; c] * A[j+1, j+2]
        end
        ## END
    end
    Q, R
end
```

```
A = Tridiagonal([1, 2, 3, 4], [1, 2, 3, 4, 5], [1, 2, 3, 4])
Q, R = bandedqr(A)
@test Q*R ≈ A
```

Problem 4.4⋆ (B) Could one redesign the above to only use IEEE operatations (addition, multiplication, square-roots,
avoiding calls `atan`, `cos`, and `sin`)?
Would it have been possible to implement this algorithm using reflections?
If so, what would be the structure of a matrix whose columns are the vectors of reflections?

SOLUTION

Yes, one could avoid using `atan`, `cos`, and `sin`. At each stage of the algorithm, we only use `atan` to compute `θ[i]` which we then use to compute Q_i^{-1} and thus Q at the end. Instead, one could simply compute Q_i^{-1} directly from t_{i-1} and $a_{i+1,i}$ and save it at each stage. An efficient implementation of $Q_1^{-1} \dots Q_n^{-1}$ would then compute Q at the end.
One could implement this using Householder Reflections as well. Interpreting the 'vectors of reflections' to mean the vectors which define each Householder reflection (padded with 0s at the front so that they are the same dimension and fit into a matrix), the matrix with these columns would be a lower bidiagonal matrix with bandwidths $(l, u) = (1, 0)$.

END

5. PLU decomposition

Problem 5.1⋆ (C) Compute the PLU decompositions for the following matrices:

$$\begin{bmatrix} 0 & 2 & 1 \\ 2 & 6 & 1 \\ 1 & 1 & 4 \end{bmatrix}, \begin{bmatrix} 1 & 2 & -1 & 0 \\ 2 & 4 & -2 & 1 \\ -3 & -5 & 6 & 1 \\ -1 & 2 & 8 & -2 \end{bmatrix}$$

SOLUTION

Part 1

Compute the PLU decomposition of,

$$\begin{bmatrix} 0 & 2 & 1 \\ 2 & 6 & 1 \\ 1 & 1 & 4 \end{bmatrix}$$

We begin by permuting the first and second row as $|2| > |0|$, hence,

$$P_1 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \qquad P_1 A = \begin{bmatrix} 2 & 6 & 1 \\ 0 & 2 & 1 \\ 1 & 1 & 4 \end{bmatrix}$$

We then choose,

$$L_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -\frac{1}{2} & 0 & 1 \end{bmatrix}, \qquad L_1 P_1 A = \begin{bmatrix} 2 & 6 & 1 \\ 0 & 2 & 1 \\ 0 & -2 & \frac{7}{2} \end{bmatrix}$$

There is no need to permute at this stage, so $P_2 = I_3$, the $3-$dimensional identity. Then we can choose,

$$L_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}, \qquad L_2P_2L_1P_1A = \begin{bmatrix} 2 & 6 & 1 \\ 0 & 2 & 1 \\ 0 & 0 & 9/2 \end{bmatrix} =: U$$

Since $P_2 = I_3$, this reduces to $L_2L_1P_1A = U \Rightarrow A = P_1^{-1}L_1^{-1}L_2^{-1}U$. Since P_1 simply permutes two rows, it is its own inverse, and $L_1^{-1}L_2^{-1}$ is simply,

$$L := L_1^{-1}L_2^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \frac{1}{2} & -1 & 1 \end{bmatrix}$$

Hence, we have $A = PLU$, where,

$$P = P_1^{-1} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \qquad L = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \frac{1}{2} & -1 & 1 \end{bmatrix}, \qquad U = \begin{bmatrix} 2 & 6 & 1 \\ 0 & 2 & 1 \\ 0 & 0 & 9/2 \end{bmatrix}$$

We can quickly check:

```
P = [0 1 0; 1 0 0; 0 0 1]
L = [1 0 0; 0 1 0; 1/2 -1 1]
U = [2 6 1; 0 2 1; 0 0 9/2]

P*L*U
```

Part 2

Find the PLU decomposition of,

$$A = \begin{bmatrix} 1 & 2 & -1 & 0 \\ 2 & 4 & -2 & 1 \\ -3 & -5 & 6 & 1 \\ -1 & 2 & 8 & -2 \end{bmatrix}$$

We see that we must start with the permutation,

$$P_1 := \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \qquad P_1A = \begin{bmatrix} -3 & -5 & 6 & 1 \\ 2 & 4 & -2 & 1 \\ 1 & 2 & -1 & 0 \\ -1 & 2 & 8 & -2 \end{bmatrix}$$

We then choose L_1,

$$L_1 := \begin{bmatrix} 1 & 0 & 0 & 0 \\ \frac{2}{3} & 1 & 0 & 0 \\ \frac{1}{3} & 0 & 1 & 0 \\ -\frac{1}{3} & 0 & 0 & 1 \end{bmatrix}, \qquad L_1P_1A = \begin{bmatrix} -3 & -5 & 6 & 1 \\ 0 & \frac{2}{3} & 2 & \frac{5}{3} \\ 0 & \frac{1}{3} & 1 & \frac{1}{3} \\ 0 & \frac{11}{3} & 6 & -\frac{7}{3} \end{bmatrix}$$

We then choose P_2,

$$P_2 := \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \qquad P_2L_1P_1A = \begin{bmatrix} -3 & -5 & 6 & 1 \\ 0 & \frac{11}{3} & 6 & -\frac{7}{3} \\ 0 & \frac{1}{3} & 1 & \frac{1}{3} \\ 0 & \frac{2}{3} & 2 & \frac{5}{3} \end{bmatrix}$$

We then choose L_2,

$$L_2 := \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -\frac{1}{11} & 1 & 0 \\ 0 & -\frac{2}{11} & 0 & 1 \end{bmatrix}, \qquad L_2P_2L_1P_1A = \begin{bmatrix} -3 & -5 & 6 & 1 \\ 0 & \frac{11}{3} & 6 & -\frac{7}{3} \\ 0 & 0 & \frac{5}{11} & \frac{6}{11} \\ 0 & 0 & \frac{10}{11} & \frac{23}{11} \end{bmatrix}$$

We then choose P_3 to swap the third and fourth rows,

$$P_3 := \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \qquad P_3L_2P_2L_1P_1A = \begin{bmatrix} -3 & -5 & 6 & 1 \\ 0 & \frac{11}{3} & 6 & -\frac{7}{3} \\ 0 & 0 & \frac{10}{11} & \frac{23}{11} \\ 0 & 0 & \frac{5}{11} & \frac{6}{11} \end{bmatrix}$$

We complete the triangularisation by choosing L_3,

$$L_3 := \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -\frac{1}{2} & 1 \end{bmatrix}, \qquad L_3P_3L_2P_2L_1P_1A = \begin{bmatrix} -3 & -5 & 6 & 1 \\ 0 & \frac{11}{3} & 6 & -\frac{7}{3} \\ 0 & 0 & \frac{10}{11} & \frac{23}{11} \\ 0 & 0 & 0 & -\frac{1}{2} \end{bmatrix}$$

We now consider,
$L_3P_3L_2P_2L_1P_1 = L_3\tilde{L}_2\tilde{L}_1P_3P_2P_1$,
where $\tilde{L}_1$ and $\tilde{L}_1$ satisfy,

$$P_3P_2L_1 = \tilde{L}_1P_3P_2$$
$$P_3L_2 = \tilde{L}_2P_3$$

These can be computed via,

$$\tilde{L}_1 = P_3P_2L_1P_2^{-1}P_3^{-1}$$
$$\tilde{L}_2 = P_3L_2P_3^{-1}$$

to obtain,

$$\tilde{L}_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -\frac{1}{3} & 1 & 0 & 0 \\ \frac{2}{3} & 0 & 1 & 0 \\ \frac{1}{3} & 0 & 0 & 1 \end{bmatrix}$$
$$\tilde{L}_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -\frac{2}{11} & 1 & 0 \\ 0 & -\frac{1}{11} & 0 & 1 \end{bmatrix}$$

This gives us,

$$L^{-1} = L_3\tilde{L}_2\tilde{L}_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -\frac{1}{3} & 1 & 0 & 0 \\ \frac{2}{3} & -\frac{2}{11} & 1 & 0 \\ \frac{1}{3} & -\frac{1}{11} & -\frac{1}{2} & 1 \end{bmatrix},$$

from which it is clear that,

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \frac{1}{3} & 1 & 0 & 0 \\ -\frac{2}{3} & \frac{2}{11} & 1 & 0 \\ -\frac{1}{3} & \frac{1}{11} & \frac{1}{2} & 1 \end{bmatrix}$$

Finally, we have that,

$$P = P_1^{-1}P_2^{-1}P_3^{-1} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix},$$

so that $A = PLU$, with,

$$P = P_1^{-1}P_2^{-1}P_3^{-1} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \quad L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \frac{1}{3} & 1 & 0 & 0 \\ -\frac{2}{3} & \frac{2}{11} & 1 & 0 \\ -\frac{1}{3} & \frac{1}{11} & \frac{1}{2} & 1 \end{bmatrix}, \quad U = \begin{bmatrix} -3 & -5 & 6 & 1 \\ 0 & \frac{11}{3} & 6 & -\frac{7}{3} \\ 0 & 0 & \frac{10}{11} & \frac{23}{11} \\ 0 & 0 & 0 & -\frac{1}{2} \end{bmatrix}$$

To check:

```
P=[0 0 0 1;0 0 1 0;1 0 0 0;0 1 0 0]
L=[1 0 0 0;1/3 1 0 0; -2/3 2/11 1 0; -1/3 1/11 1/2 1]
U=[-3 -5 6 1;0 11/3 6 -7/3;0 0 10/11 23/11;0 0 0 -1/2]
P*L*U
```

END

MATH50003 Numerical Analysis: Problem Sheet 5

This problem sheet explores positive definite matrices,
Cholesky decompositions, matrix norms, and the singular value decomposition.

Questions marked with a ⋆ are meant to be completed without using a computer.
Problems are denoted A/B/C to indicate their difficulty.

```
using LinearAlgebra, Plots, Test
```

1. Positive definite matrices and Cholesky decompositions

Problem 1.1⋆ (C) Use the Cholesky decomposition to determine
which of the following matrices are symmetric positive definite:

$$\begin{bmatrix} 1 & -1 \\ -1 & 3 \end{bmatrix}, \begin{bmatrix} 1 & 2 & 2 \\ 2 & 1 & 2 \\ 2 & 2 & 1 \end{bmatrix}, \begin{bmatrix} 3 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 5 \end{bmatrix}, \begin{bmatrix} 4 & 2 & 2 & 1 \\ 2 & 4 & 2 & 2 \\ 2 & 2 & 4 & 2 \\ 1 & 2 & 2 & 4 \end{bmatrix}$$

SOLUTION

A matrix is symmetric positive definite (SPD) if and only if it has a Cholesky decomposition, so the task here is really just to compute Cholesky decompositions (by hand). Since our goal is to tell if the Cholesky decompositions exist, we do not have to compute L_k's. We only need to see if the decomposition process can keep to the end.

Matrix 1

$$A_0 = \begin{bmatrix} 1 & -1 \\ -1 & 3 \end{bmatrix}$$

$A_1 = 3 - \frac{(-1)\times(-1)}{1} > 0$, so Matrix 1 is SPD.

Matrix 2

$$A_0 = \begin{bmatrix} 1 & 2 & 2 \\ 2 & 1 & 2 \\ 2 & 2 & 1 \end{bmatrix}$$

$$A_1 = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} - \begin{bmatrix} 2 \\ 2 \end{bmatrix} \begin{bmatrix} 2 & 2 \end{bmatrix} = \begin{bmatrix} -3 & -2 \\ -2 & -3 \end{bmatrix}$$

$A_1[1,1] < 0$, so Matrix 2 is not SPD.

Matrix 3

$$A_0 = \begin{bmatrix} 3 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 5 \end{bmatrix}$$

$$A_1 = \begin{bmatrix} 4 & 2 \\ 2 & 5 \end{bmatrix} - \frac{1}{3}\begin{bmatrix} 2 \\ 1 \end{bmatrix} \begin{bmatrix} 2 & 1 \end{bmatrix} = \frac{1}{3}\begin{bmatrix} 8 & 4 \\ 4 & 13 \end{bmatrix}$$

$3A_2 = 13 - \frac{4\times4}{8} > 0$, so Matrix 3 is SPD.

Matrix 4

$$A_0 = \begin{bmatrix} 4 & 2 & 2 & 1 \\ 2 & 4 & 2 & 2 \\ 2 & 2 & 4 & 2 \\ 1 & 2 & 2 & 4 \end{bmatrix}$$

$$A_1 = \begin{bmatrix} 4 & 2 & 2 \\ 2 & 4 & 2 \\ 2 & 2 & 4 \end{bmatrix} - \frac{1}{4}\begin{bmatrix} 2 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} 2 & 2 & 1 \end{bmatrix} = \frac{1}{4}\begin{bmatrix} 12 & 4 & 6 \\ 4 & 12 & 6 \\ 6 & 6 & 15 \end{bmatrix}$$

$$4A_2 = \begin{bmatrix} 12 & 6 \\ 6 & 15 \end{bmatrix} - \frac{1}{12}\begin{bmatrix} 4 \\ 6 \end{bmatrix} \begin{bmatrix} 4 & 6 \end{bmatrix} = \frac{4}{3}\begin{bmatrix} 8 & 3 \\ 3 & 9 \end{bmatrix}$$

$3A_3 = 9 - \frac{3\times3}{8} > 0$, so Matrix 4 is SPD.

We can check that we did this correctly by running the following in Julia:

```
cholesky([1 -1; -1 3])
```

```
# this throws an error when uncommented and run because the matrix is not SPD
# cholesky([1 2 2; 2 1 2; 2 2 1])
```

```
cholesky([3 2 1; 2 4 2; 1 2 5])
```

```
cholesky([4 2 2 1; 2 4 2 2; 2 2 4 2; 1 2 2 4])
```

END

Problem 1.2⋆ (B) Recall that an inner product $\langle \mathbf{x}, \mathbf{y} \rangle$ on $\mathbb{R}^n$ over the reals $\mathbb{R}$ satisfies, for all $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{R}$ and $a, b \in \mathbb{R}$:

1. Symmetry: $\langle \mathbf{x}, \mathbf{y} \rangle = \langle \mathbf{y}, \mathbf{x} \rangle$
2. Linearity: $\langle a\mathbf{x} + b\mathbf{y}, \mathbf{z} \rangle = a\langle \mathbf{x}, \mathbf{z} \rangle + b\langle \mathbf{y}, \mathbf{z} \rangle$
3. Posive-definite: $\langle \mathbf{x}, \mathbf{x} \rangle > 0, x \neq 0$

Prove that $\langle \mathbf{x}, \mathbf{y} \rangle$ is an inner product if and only if

$$\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^\top K \mathbf{y}$$

where K is a symmetric positive definite matrix.

SOLUTION

We begin by showing that $\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^\top K \mathbf{y}$ with K spd defines an inner product. To do this we simply verify the three properties: For symmetry, we find
$\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^\top K \mathbf{y} = \mathbf{x} \cdot (K\mathbf{y}) = (K\mathbf{y}) \cdot \mathbf{x}$
$= (K\mathbf{y})^\top \mathbf{x} = \mathbf{y}^\top K^\top \mathbf{x} = \mathbf{y}^\top K \mathbf{x} = \langle \mathbf{y}, \mathbf{x} \rangle.$
For linearity:
$\langle a\mathbf{x} + b\mathbf{y}, \mathbf{z} \rangle = (a\mathbf{x} + b\mathbf{y})^\top K \mathbf{z} = (a\mathbf{x}^\top + b\mathbf{y}^\top) K \mathbf{z}$
$= a\mathbf{x}^\top K \mathbf{z} + b\mathbf{y}^\top K \mathbf{z} = a\langle \mathbf{x}, \mathbf{z} \rangle + b\langle \mathbf{y}, \mathbf{z} \rangle.$
Positive-definiteness of the matrix K immediately yields $\langle \mathbf{x}, \mathbf{x} \rangle = \mathbf{x}^\top K \mathbf{x} > 0$. Now we turn to the converse result, i.e. that there exists a symmetric positive definite matrix K for any inner product $\langle \mathbf{x}, \mathbf{y} \rangle$ such that it can be written as $\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^\top K \mathbf{y}$. Define the entries of K by $K_{ij} = \langle e_i, e_j \rangle$ where e_j is the j-th standard basis vector. Note that by linearity of the inner product any inner product on $\mathbb{R}^n$ can be written as $\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{k=0}^n \sum_{l=0}^n x_k y_l \langle e_k, e_l \rangle$ by linearity. But with the elements of K defined as above this is precisely
$\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{k=0}^n \sum_{l=0}^n x_k K_{kl} y_l = \mathbf{x}^\top K \mathbf{y}.$
What remains is to show that this K is symmetric positive definite. Symmetry is an immediate conseqence of the symmetry of its elements, i.e. $K_{ij} = \langle e_i, e_j \rangle = \langle e_j, e_i \rangle = K_{ji}$. Finally, positive definiteness follows from the positive definiteness of the inner product $\langle \mathbf{x}, \mathbf{x} \rangle > 0$ with $\langle \mathbf{x}, \mathbf{x} \rangle = \mathbf{x}^\top K \mathbf{x}$.

END

Problem 1.3⋆ (A) Show that a matrix is symmetric positive definite if and only if it has a Cholesky decomposition of the form

$$A = UU^\top$$

where U is upper triangular with positive entries on the diagonal.

SOLUTION

We didn't discuss this but note that because a symmetric positive definite matrix has stricly positive eigenvalues: for a normalised
eigenvector we have

$$\lambda = \lambda \mathbf{v}^\top \mathbf{v} = \mathbf{v}^\top K \mathbf{v} > 0.$$

Thus they are always invertible. Then note that any such matrix has a Cholesky decomposition of standard form $A = LL^\top$ where L is lower triangular. The inverse of this standard form Cholesky decomposition is then $A^{-1} = L^{-T}L^{-1}$, which is of the desired form since L is lower triangular and L^{-T} is upper triangular. The positive entries on the diagonal follow directly because this is the case for the Cholesky decomposition factors of the original matrix. Thus, since all symmetric positive definite matrices can be written as the inverses of a symmetric positive definite matrix, this shows that they all have a decomposition $A = UU^\top$ (using the Cholesky factors of its inverse).

Alternatively, we can replicate the procedure of computing the Cholesky decomposition beginning in the bottom right
instead of the top left. Write:

$$A = \begin{bmatrix} K & \mathbf{v} \\ \mathbf{v}^\top & \alpha \end{bmatrix} = \underbrace{\begin{bmatrix} I & \frac{\mathbf{v}}{\sqrt{\alpha}} \\ & \sqrt{\alpha} \end{bmatrix}}_{U_1} \begin{bmatrix} K - \frac{\mathbf{v}\mathbf{v}^\top}{\alpha} & \\ & 1 \end{bmatrix} \underbrace{\begin{bmatrix} I & \\ \frac{\mathbf{v}^\top}{\sqrt{\alpha}} & \sqrt{\alpha} \end{bmatrix}}_{U_1^\top}$$

The induction proceeds as in the lower triangular case.

END

Problem 1.4⋆ (A) Prove that the following $n \times n$ matrix is symmetric positive definite
for any n:

$$\Delta_n := \begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & 2 & \ddots & \\ & & \ddots & \ddots & -1 \\ & & & -1 & 2 \end{bmatrix}$$

Deduce its two Cholesky decompositions: $\Delta_n = L_n L_n^\top = U_n U_n^\top$ where
L_n is lower triangular and U_n is upper triangular.

SOLUTION

We first prove that L_n is lower bidiagonal by induction. Let A be a tridiagonal SPD matrix:

$$A = \left[\begin{array}{c|ccc} \alpha & \beta & 0 & \cdots \\ \hline \beta & & & \\ 0 & & K & \\ \vdots & & & \end{array}\right]$$

where K is again tridiagonal. Denote the Cholesky decomposition of A by $A = LL^\top$. Recalling the proof of the *Theorem (Cholesky & SPD)* from the lecture, we can write

$$L = \left[\begin{array}{c|c} \sqrt{\alpha} & 0 \\ \hline \frac{\beta}{\sqrt{\alpha}} & \\ 0 & \tilde{L} \\ \vdots & \end{array}\right]$$

where $\tilde{L}$ satisfies $\tilde{L}\tilde{L}^\top = K - \begin{bmatrix} \beta^2/\alpha & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}$ which is a tridiagonal matrix smaller than A. Since L is lower bidiagonal when A is 1×1, we know by induction that L is lower bidiagonal for A of any size.

Once we know that L_n is lower bidiagonal, we can write it as

$$L_n = \begin{bmatrix} a_1 & & & \\ b_1 & \ddots & & \\ & \ddots & \ddots & \\ & & b_{n-1} & a_n \end{bmatrix}$$

which we substitute into $\Delta_n = L_n L_n^\top$ to get

$$\begin{cases} a_1 = \sqrt{2} \\ a_k b_k = -1 \\ b_k^2 + a_{k+1}^2 = 2 \end{cases}$$

for $k = 1, \dots, n-1$.

Now we solve the recurrence. Substituting the second equation into the third one:
$\frac{1}{a_k^2} + a_{k+1}^2 = 2.$
Let $c_k = a_k^2$:
$\frac{1}{c_k} + c_{k+1} = 2.$
Consider the fixing point of this recurrence: $\frac{1}{x} + x = 2 \Longrightarrow (x-1)^2 = 0$ has the double root $x = 1$ which hints us to consider $\frac{1}{c_k - 1}$. In fact, the recurrence is equivalent to
$\frac{1}{c_{k+1}-1} = \frac{1}{c_k - 1} + 1.$
Recalling that $c_1 = 2$, we know that $\frac{1}{c_k - 1} = k$. As a result, $c_k = (k+1)/k$, $a_k = \sqrt{(k+1)/k}$ and $b_k = -\sqrt{k/(k+1)}$, hence we know L_n.

We can apply the same process to U_n, but this is a special case since flipping Δ_n horizontally and vertically gives itself: $P\Delta_n P^\top = \Delta_n$ where

$$P = \begin{bmatrix} & & 1 \\ & \iddots & \\ 1 & & \end{bmatrix}$$

is the permutation that reverses a vector.
So we can also flip L_n to get U_n:

$$U_n = P L_n P$$

so that $U_n U_n^\top = P L_n P P L_n^\top P = P \Delta_n P = \Delta_n$.

Alternatively one can use the procedure from Problem 1.3. That is, write:

$$\Delta_n = \begin{bmatrix} \Delta_{n-1} & -\mathbf{e}_n \\ -\mathbf{e}_n^\top & 2 \end{bmatrix} = \underbrace{\begin{bmatrix} I & \frac{-\mathbf{e}_n}{\sqrt{2}} \\ & \sqrt{2} \end{bmatrix}}_{U_1} \begin{bmatrix} \Delta_{n-1} - \frac{\mathbf{e}_n \mathbf{e}_n^\top}{2} & \\ & 1 \end{bmatrix} \underbrace{\begin{bmatrix} I & \\ \frac{\mathbf{v}^\top}{\sqrt{\alpha}} & \sqrt{\alpha} \end{bmatrix}}_{U_1^\top}$$

Continuing proceeds as above.

END

Problem 1.5 (B) `SymTridiagonal(dv, eu)` is a type for representing symmetric tridiagonal matrices (that is, `SymTridiagonal(dv, ev) == Tridiagonal(ev, dv, ev)`). Complete the following implementation of `cholesky` to return a `Bidiagonal` cholesky factor in $O(n)$ operations, and check your result
compared to your solution of Problem 1.3 for `n = 1_000_000`.

```
import LinearAlgebra: cholesky

# return a Bidiagonal L such that L'L == A (up to machine precision)
cholesky(A::SymTridiagonal) = cholesky!(copy(A))

# return a Bidiagonal L such that L'L == A (up to machine precision)
# You are allowed to change A
function cholesky!(A::SymTridiagonal)
    d = A.dv # diagonal entries of A
    u = A.ev # sub/super-diagonal entries of A
    T = float(eltype(A)) # return type, make float in case A has Ints
    n = length(d)
    ld = zeros(T, n) # diagonal entries of L
    ll = zeros(T, n-1) # sub-diagonal entries of L

    ## SOLUTION
    ld[1]=sqrt(d[1])
    for k=1:n-1
        ll[k]=u[k]/ld[k]
        ld[k+1]=sqrt(d[k+1]-ll[k]^2)
    end
    ## END

    Bidiagonal(ld, ll, :L)
end

n = 1000
A = SymTridiagonal(2*ones(n),-ones(n-1))
L = cholesky(A)
@test L ≈ cholesky(Matrix(A)).L
```

2. Matrix norms

Problem 2.1⋆ (B) Prove the following:

$$\|A\|_\infty = \max_k \|A[k,:]\|_1$$
$$\|A\|_{1\to\infty} = \|\mathrm{vec}(A)\|_\infty = \max_{kj} |a_{kj}|$$

SOLUTION

Step 1. upper bounds

$$\|A\mathbf{x}\|_\infty = \max_k \left|\sum_j a_{kj}x_j\right| \le \max_k \sum_j |a_{kj}x_j| \le \begin{cases} \max_j |x_j| \max_k \sum_j |a_{kj}| = \|\mathbf{x}\|_\infty \max_k \|A[k,:]\|_1 \\ \max_{kj} |a_{kj}| \sum_j |x_j| = \|\mathbf{x}\|_1 \|\mathrm{vec}(A)\|_\infty \end{cases}$$

Step 2.1. meeting the upper bound ($\|A\|_{1\to\infty}$)

Let a_{lm} be the entry of A with maximum absolute value. Let $\mathbf{x} = \mathbf{e}_m$, then
$\|A\mathbf{x}\|_\infty = \max_k \left|\sum_j a_{kj}x_j\right| = \max_k |a_{km}| = |a_{lm}|$
and
$\|\mathbf{x}\|_1 \|\mathrm{vec}(A)\|_\infty = 1 \cdot |a_{lm}|.$

Step 2.2. meeting the upper bound ($\|A\|_\infty$)

Let $A[n,:]$ be the row of A with maximum 1-norm. Let $\mathbf{x} = (\mathrm{sign.}\,(A[n,:]))^\top$, then
$\left|\sum_j a_{kj}x_j\right| \begin{cases} = \sum_j |a_{kj}| = \|A[k,:]\|_1 & k = n \\ \le \sum_j |a_{kj}| = \|A[k,:]\|_1 & k \ne n' \end{cases}$ so
$\|A\mathbf{x}\|_\infty = \max_k \left|\sum_j a_{kj}x_j\right| = \max_k \|A[k,:]\|_1$
while
$\|\mathbf{x}\|_\infty \max_k \|A[k,:]\|_1 = 1 \cdot \max_k \|A[k,:]\|_1.$

Conclusion

In both cases, equality can hold, so the upper bounds are actually maxima.

END

Problem 2.2⋆ (B) For a rank-1 matrix $A = \mathbf{x}\mathbf{y}^\top$ prove that

$$\|A\|_2 = \|\mathbf{x}\|_2\|\mathbf{y}\|_2.$$

Hint: use the Cauchy–Schwartz inequality.

SOLUTION

$\|A\mathbf{z}\|_2 = \|\mathbf{x}\mathbf{y}^\top\mathbf{z}\|_2 = |\mathbf{y}^\top\mathbf{z}|\|\mathbf{x}\|_2,$
so it remains to prove that $\|\mathbf{y}\|_2 = \sup_{\mathbf{z}} \frac{|\mathbf{y}^\top\mathbf{z}|}{\|\mathbf{z}\|_2}$.

By Cauchy-Schwartz inequality,
$|\mathbf{y}^\top\mathbf{z}| = |(\mathbf{y},\mathbf{z})| \le \|\mathbf{y}\|_2\|\mathbf{z}\|_2$
with the two sides being equal when $\mathbf{y}$ and $\mathbf{z}$ are linearly dependent, in which case the bound is tight.

END

Problem 2.3⋆ (B) Show for any orthogonal matrix $Q \in \mathbb{R}^m$ and
matrix $A \in \mathbb{R}^{m\times n}$ that

$$\|QA\|_F = \|A\|_F$$

by first showing that $\|A\|_F = \sqrt{\mathrm{tr}(A^\top A)}$ using the
trace of an $m\times m$ matrix:

$$\mathrm{tr}(A) = a_{11} + a_{22} + \cdots + a_{mm}.$$

SOLUTION

$\mathrm{tr}(A^\top A) = \sum_k (A^\top A)[k,k] = \sum_k \sum_j A^\top[k,j]A[j,k] = \sum_k \sum_j A[j,k]^2 = \|A\|_F^2.$

On the other hand,
$\mathrm{tr}(A^\top A) = \mathrm{tr}(A^\top Q^\top QA) = \mathrm{tr}((QA)^\top(QA)) = \|QA\|_F^2,$
so $\|QA\|_F = \|A\|_F$.

END

3. Singular value decomposition

Problem 3.1⋆ (B) Show that $\|A\|_2 \leq \|A\|_F \leq \sqrt{r}\|A\|_2$ where r is the rank of A.

SOLUTION

From Problem 2.3 use the fact that $\|A\|_F = \sqrt{\mathrm{tr}(A^\top A)}$, where $A \in \mathbb{R}^{m\times n}$.

Hence,

$$\|A\|_F^2 = \mathrm{tr}(A^\top A) = \sigma_1^2 + \ldots + \sigma_m^2$$

where $\sigma_1 \geq \ldots \geq \sigma_n \geq 0$ are the singular values of A and σ_i^2 are the eigenvalues of $A^\top A$

Knowing that $\|A\|_2^2 = \sigma_1^2$ we have $\|A\|_2^2 \leq \|A\|_F^2$

Moreover, since if the rank of A is r we have that $\sigma_{r+1} = \ldots = \sigma_m = 0$ and we also know $\sigma_1 \geq \ldots \geq \sigma_n \geq 0$, we have that

$$\|A\|_F^2 = \sigma_1^2 + \ldots + \sigma_m^2 = \sigma_1^2 + \ldots + \sigma_r^2 \leq r\sigma_1^2 = r\|A\|_2^2$$

Hence,

$$\|A\|_2 \leq \|A\|_F \leq \sqrt{r}\|A\|_2.$$

END

Problem 3.2 (A) Consider functions sampled on a $(n+1)\times(n+1)$ 2D grid $(x_k, y_j) = (k/n, j/n)$ where $k, j = 0, \ldots, n$.
For $n = 100$, what is the lowest rank r such that
the best rank-r approximation to the samples
that is accurate to within 10^{-5} accuracy for the following functions:

$$(x+2y)^2, \cos(\sin x \mathrm{e}^y), 1/(x+y+1), \mathrm{sign}(x-y)$$

For which examples does the answer change when $n = 1000$?

SOLUTION

```
#define functions
f₁(x,y) = (x + 2 * y) ^ 2
f₂(x,y) = cos(sin(x)*exp(y))
f₃(x,y) = 1/(x + y + 1)
f₄(x,y) = sign(x-y)

#define n and error goal
error = 1e-5

#helper function to compute nxn samples
function samples(f, n)
   x = y = range(0, 1; length=n)
   return f.(x,y')
end
```

```
function find_min_rank(f, n, ϵ)
    F = samples(f,n)
    U,σ,V = svd(F)
    for k=1:n
        Σ_k = Diagonal(σ[1:k])
        U_k = U[:,1:k]
        V_k = V[:,1:k]
        if norm(U_k * Σ_k * V_k' - F) <= ϵ
            return k
        end
    end
end

n=100
println("Error ≤ ", error, " with n = ", n)
println("Rank for f₁ = ", find_min_rank(f₁, n, error))
println("Rank for f₂ = ", find_min_rank(f₂, n, error))
println("Rank for f₃ = ", find_min_rank(f₃, n, error))
println("Rank for f₄ = ", find_min_rank(f₄, n, error))

n=1000
println("Error ≤ ", error, " with n = ", n)
println("Rank for f₁ = ", find_min_rank(f₁, n, error))
println("Rank for f₂ = ", find_min_rank(f₂, n, error))
println("Rank for f₃ = ", find_min_rank(f₃, n, error))
println("Rank for f₄ = ", find_min_rank(f₄, n, error))
```

END

Problem 3.3⋆ (B) For $A \in \mathbb{R}^{m\times n}$ define the *pseudo-inverse*:

$$A^+ := V\Sigma^{-1}U^\top.$$

Show that it satisfies the *Moore-Penrose conditions*:

1. $AA^+A = A$
2. $A^+AA^+ = A^+$
3. $(AA^+)^\top = AA^+$ and $(A^+A)^\top = A^+A$

SOLUTION

Let $A = U\Sigma V^\top$ and $A^+ := V\Sigma^{-1}U^\top$, where $U \in \mathbb{R}^{m\times r}$ and $V \in \mathbb{R}^{n\times r}$. Note that $U^\top U = I_m$ and $V^\top V = I_r$.

1. We have

$$AA^+A = U\Sigma V^\top V\Sigma^{-1}U^\top U\Sigma V^\top = U\Sigma\Sigma^{-1}\Sigma V^\top = U\Sigma V^\top = A$$

2. Moreover,

$$A^+AA^+ = V\Sigma^{-1}U^\top U\Sigma V^\top V\Sigma^{-1}U^\top = V\Sigma^{-1}\Sigma\Sigma^{-1}U^\top = V\Sigma^{-1}U^\top = A^+$$

3.

$$\begin{aligned}(AA^+)^\top &= (A^+)^\top A^\top = U\Sigma^{-1}V^\top V\Sigma U^\top = UU^\top = U\Sigma V^\top V\Sigma^{-1}U^\top = AA^+\\ (A^+A)^\top &= A^\top(A^+)^\top = V\Sigma U^\top U\Sigma^{-1}V^\top = VV^\top = V\Sigma^{-1}U^\top U\Sigma V^\top = A^+A\end{aligned}$$

END

Problem 3.4⋆ (A) Show for $A \in \mathbb{R}^{m\times n}$ with $m \geq n$ and $\top$ rank that $\mathbf{x} = A^+\mathbf{b}$ is the least squares solution, i.e., minimises $\|A\mathbf{x} - \mathbf{b}\|_2$.
Hint: extend U in the SVD to be a square orthogonal matrix.

SOLUTION

The proof mimics that of the QR decomposition. Write $A = U\Sigma V^\top$ and let

$$\mathcal{U} = \begin{bmatrix} U & K \end{bmatrix}$$

so that $\mathcal{U}$ is orthogonal. We use the fact orthogonal matrices do not change norms:

$$\|A\mathbf{x} - \mathbf{b}\|_2^2 = \|U\Sigma V^\top \mathbf{x} - \mathbf{b}\|_2^2 = \|\mathcal{U}^\top U\Sigma V^\top \mathbf{x} - \mathcal{U}^\top \mathbf{b}\|_2^2 = \|\underbrace{\begin{bmatrix} I_m \\ O \end{bmatrix}}_{\in\mathbb{R}^{m\times n}} \Sigma V^\top \mathbf{x} - \begin{bmatrix} U^\top \\ K^\top \end{bmatrix}\mathbf{b}\|_2^2$$

$$= \|\Sigma V^\top \mathbf{x} - U^\top \mathbf{b}\|_2^2 + \|K^\top \mathbf{b}\|^2$$

The second term is independent of $\mathbf{x}$. The first term is minimised when zero:

$$\|\Sigma V^\top \mathbf{x} - U^\top \mathbf{b}\|_2 = \|\Sigma V^\top V\Sigma^{-1}U^\top \mathbf{b} - U^\top \mathbf{b}\|_2 = 0$$

END

Problem 3.5⋆ (A)
If $A \in \mathbb{R}^{m\times n}$ has a non-empty kernel there are multiple solutions to the least squares problem as
we can add any element of the kernel. Show that $\mathbf{x} = A^+\mathbf{b}$ gives the least squares solution such that $\|\mathbf{x}\|_2$ is minimised.

SOLUTION

Let $\mathbf{x} = A^+ b$ and let $\mathbf{x} + \mathbf{k}$ to be another solution i.e.

$$\|A\mathbf{x} - b\| = \|A(\mathbf{x} + \mathbf{k}) - b\|$$

Following the previous part we deduce:

$$\Sigma V^\top(\mathbf{x} + \mathbf{k}) = U^\top \mathbf{b} \Rightarrow V^\top \mathbf{k} = 0$$

As $\mathbf{x} = V\mathbf{c}$ lies in the span of the columns of V we have
$\mathbf{x}^\top \mathbf{k} = 0$. Thus

$$\|\mathbf{x} + \mathbf{k}\|^2 = \|\mathbf{x}\|^2 + \|\mathbf{k}\|^2$$

which is minimised when $\mathbf{k} = 0$.

END

MATH50003 Numerical Analysis: Problem Sheet 6

This problem sheet explores condition numbers, indefinite integration,
and Euler's method.

Questions marked with a ⋆ are meant to be completed without using a computer.
Problems are denoted A/B/C to indicate their difficulty.

```
using LinearAlgebra, Plots
```

1. Condition numbers

Problem 1.1⋆ (B) Prove that, if $|\epsilon_i| \leq \epsilon$ and $n\epsilon < 1$, then

$$\prod_{k=1}^n (1+\epsilon_i) = 1 + \theta_n$$

for some constant θ_n satisfying $|\theta_n| \leq \frac{n\epsilon}{1-n\epsilon}$.

SOLUTION

$\prod_{k=1}^n (1+\epsilon_i) \leq (1+\epsilon)^n = \sum_{k=0}^n \binom{n}{k} \epsilon^k \leq 1 + \sum_{k=1}^n n^k \epsilon^k \leq 1 + \sum_{k=1}^\infty n^k \epsilon^k = 1 + \frac{n\epsilon}{1-n\epsilon}$.
$\prod_{k=1}^n (1+\epsilon_i) \geq (1-\epsilon)^n = \sum_{k=0}^n \binom{n}{k} (-\epsilon)^k \geq 1 - \sum_{k=1}^n n^k \epsilon^k \geq 1 - \sum_{k=1}^\infty n^k \epsilon^k = 1 - \frac{n\epsilon}{1-n\epsilon}$.

Problem 1.2⋆ (B) Let $A, B \in \mathbb{R}^{m\times n}$. Prove that if the columns satisfy $\|\mathbf{a}_j\|_2 \leq \|\mathbf{b}_j\|_2$ then $\|A\|_F \leq \|B\|_F$ and $\|A\|_2 \leq \sqrt{\operatorname{rank}(B)}\|B\|_2$.

SOLUTION

Recalling from ***Problem Sheet 5*** - *Problem 2.3** - *SOLUTION*, we know that
$\|A\|_F = \sqrt{\sum_{k,j} A[k,j]^2} = \sqrt{\sum_j \|\mathbf{a}_j\|_2^2}$ and $\|B\|_F = \sqrt{\sum_j \|\mathbf{b}_j\|_2^2}$.
Since $\|\mathbf{a}_j\|_2 \leq \|\mathbf{b}_j\|_2$, we have $\|A\|_F \leq \|B\|_F$.

Recalling from ***Problem Sheet 5*** - *Problem 3.1**, we have
$\|A\|_2 \leq \|A\|_F \leq \|B\|_F \leq \sqrt{\operatorname{rank}(B)}\|B\|_2$.

Problem 1.3⋆ (C) Compute the 1-norm, 2-norm, and ∞-norm condition numbers for the following matrices:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \begin{bmatrix} 1/3 & 1/5 \\ 0 & 1/7 \end{bmatrix}, \begin{bmatrix} 1 & & & \\ & 1/2 & & \\ & & \cdots & \\ & & & 1/2^n \end{bmatrix}$$

(Hint: recall that the singular values of a matrix A are the square roots of the eigenvalues of the Gram matrix $A^\top A$.)

SOLUTION

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \qquad A^{-1} = -\frac{1}{2}\begin{bmatrix} 4 & -2 \\ -3 & 1 \end{bmatrix}$$
$$B = \begin{bmatrix} 1/3 & 1/5 \\ 0 & 1/7 \end{bmatrix}, \qquad B^{-1} = 21\begin{bmatrix} 1/7 & -1/5 \\ 0 & 1/3 \end{bmatrix}$$

$\|A\|_1 = 6$, $\|A^{-1}\|_1 = 7/2$, so $\kappa_1(A) = 21$.

$\|A\|_\infty = 7$, $\|A^{-1}\|_\infty = 3$, so $\kappa_\infty(A) = 21$.

$\|B\|_1 = 12/35$, $\|B^{-1}\|_1 = 21 \times 8/15 = 56/5$, so $\kappa_1(B) = 96/25$.

$\|B\|_\infty = 8/15$, $\|B^{-1}\|_\infty = 21 \times 12/35$, so $\kappa_\infty(B) = 96/25$

Finally, for the 2-norms:
$\kappa_2(A)$:
For $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$, we have that the singular values are the $\sigma_1 = \sqrt{\lambda_1}, \sigma_2 = \sqrt{\lambda_2}$, where λ_1 and λ_2 are the eigenvalues of $A^T A$.

$$A^T A = \begin{bmatrix} 10 & 14 \\ 14 & 20 \end{bmatrix},$$

so an eigenvalue λ of $A^T A$ must satisfy,

$$\begin{aligned} (10 - \lambda)(20 - \lambda) - 196 = 0 \\ \Leftrightarrow \lambda = 15 \pm \sqrt{221}. \end{aligned}$$

The larger eigenvalue corresponds to σ_1, so $\sigma_1 = \sqrt{15 + \sqrt{221}}$, and the smaller corresponds to σ_2, so $\sigma_2 = \sqrt{15 - \sqrt{221}}$. Finally, we have $\|A\|_2 = \sigma_1$, $\|A^{-1}\|_2 = 1/\sigma_2$, and so $\kappa_2(A) = \sqrt{\frac{15+\sqrt{221}}{15-\sqrt{221}}}$.

$\kappa_2(B)$:
For

$$B = \begin{bmatrix} 1/3 & 1/5 \\ 0 & 1/7 \end{bmatrix},$$

we have that the singular values are the $\sigma_1 = \sqrt{\lambda_1}, \sigma_2 = \sqrt{\lambda_2}$, where λ_1 and λ_2 are the eigenvalues of $A^T A$.

$$A^T A = \begin{bmatrix} 1/9 & 1/15 \\ 1/15 & \frac{74}{5^2 7^2} \end{bmatrix}.$$

An eigenvalue λ must satisfy:
\begin{align}
(1/9 - \lambda)\left(\frac{74}{5^27^2}-\lambda\right) - \frac{1}{225} = 0 \\
\Leftrightarrow \lambda = \frac{1891 \pm 29\sqrt{2941}}{22050}.
\end{align}
With the same logic as above, we can then deduce that $\|B\|_2 = \sqrt{\frac{1891+29\sqrt{2941}}{22050}}$ and $\|B^{-1}\|_2 \sqrt{\frac{22050}{1891-29\sqrt{2941}}}$ so that,

$$\kappa_2(B) = \sqrt{\frac{1891 + 29\sqrt{2941}}{1891 - 29\sqrt{2941}}}$$

For,

$$A_n = \begin{bmatrix} 1 & & & \\ & 1/2 & & \\ & & \ddots & \\ & & & 1/2^n \end{bmatrix}, \quad A_n^{-1} = \begin{bmatrix} 1 & & & \\ & 2 & & \\ & & \ddots & \\ & & & 2^n \end{bmatrix}$$

It is clear that $\|A_n\|_1 = \|A_n\|_\infty = 1$, and $\|A_n^{-1}\|_1 = \|A_n^{-1}\|_\infty = 2^n$, so $\kappa_1(A_n) = \kappa_\infty(A) = 2^n$.
Morover, we can clearly see the singular values $\sigma_1 = 1, \sigma_2 = 1/2, \ldots, \sigma_{n+1} = 1/2^n$. So
$\|A_n\|_2 = 1$, $\|A_n^{-1}\|_2 = 2^n$, $\kappa_2(A_n) = 2^n$

Problem 1.4 (B)

State a bound on the relative error on $A\mathbf{v}$ for $\|\mathbf{v}\|_2 = 1$ for the following matrices:

$$\begin{bmatrix} 1/3 & 1/5 \\ 0 & 1/10^3 \end{bmatrix}, \begin{bmatrix} 1 & & & \\ & 1/2 & & \\ & & \cdots & \\ & & & 1/2^{10} \end{bmatrix}$$

Compute the relative error in computing $A\mathbf{v}$ (using `big` for a high-precision version to compare against) where $\mathbf{v}$ is the last column of V in the SVD $A = U\Sigma V^\top$, computed using the `svd` command with `Float64` inputs. How does the error compare to the predicted error bound?

SOLUTION

The Theorem (relative-error for matrix vector) tells us that,

$$\frac{\|\delta A\mathbf{x}\|}{\|A\mathbf{x}\|} \le \kappa(A)\epsilon,$$

if the relative pertubation error $\|\delta A\| = \|A\|\epsilon$. For the 2-norm, we have,

$$\|\delta A\|_2 \le \underbrace{\frac{\sqrt{\min(m,n)}n\epsilon_m}{2 - n\epsilon_m}}_{\epsilon} \|A\|_2.$$

The condition number of the first matrix is 453.33 (see code below to compute that), and ϵ defined above is $\frac{2\sqrt{2}\epsilon_m}{2-2\epsilon_m} = 3.14 \cdot 10^{-16}$, so the bound on the relative error is:
$1.42 \cdot 10^{-13}$.
The condition number of the second matrix is 2^{10} by the question above, and ϵ defined above is $\frac{10\sqrt{10}\epsilon_m}{2-10\epsilon_m} = 7.02 \cdot 10^{-16}$, the bound on the relative error in this case is then:

$$7.19 \cdot 10^{-13}$$

```
using LinearAlgebra

A = [1/3 1/5; 0 1/1000]
U,σ,V = svd(A)
κ = σ[1]/σ[end]
v = V[:,end]
```

```
A_big = [big(1)/3 big(1)/5; 0 big(1)/1000]
```

```
norm(A_big*v - A*v, 2)/norm(A_big*v, 2)
```

```
2*sqrt(2)*eps()/(2-2*eps())* κ
```

```
B = diagm( 2.0 .^(0:-1:-10))
U,σ,V = svd(B)
κ = σ[1]/σ[end]
v = V[:,end]
```

```
B*v
```

Note, this is exact so the relative error is 0, within the upper bound.

```
10*sqrt(10)*eps()/(10-10*eps()) * 2^(10)
```

2. Indefinite integration

Problem 2.1 (B) Implement backward differences to approximate indefinite-integration. How does the error compare to forward and mid-point versions for $f(x) = \cos x$ on the interval $[0,1]$? Use the method to approximate the integrals of

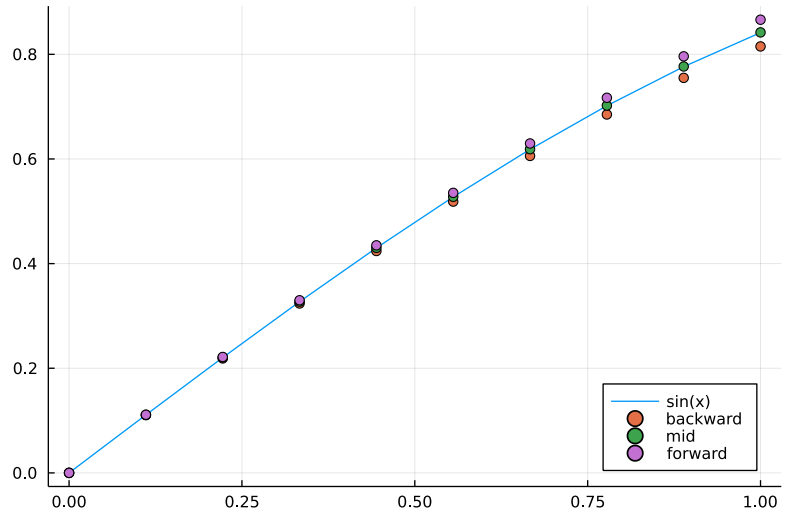
$$\exp\left(\exp x \cos x + \sin x\right), \prod_{k=1}^{1000} \left(\frac{x}{k} - 1\right), \text{ and } f_{1000}^{\mathrm{s}}(x)$$

to 3 digits, where $f_{1000}^{\mathrm{s}}(x)$ was defined in PS2.

SOLUTION

We can implement the backward difference solution as follows:

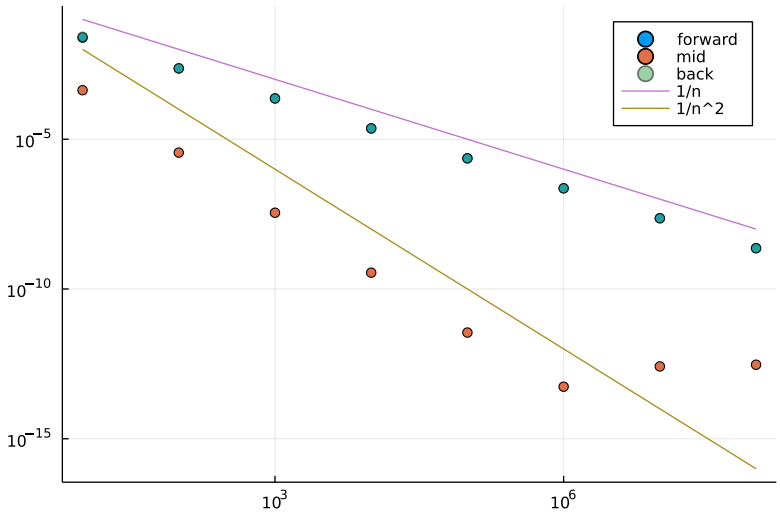
```
c = 0 # u(0) = 0
f = x -> cos(x)
n = 10

x = range(0,1;length=n)
h=step(x)
A = Bidiagonal([1; fill(1/h, n-1)], fill(-1/h, n-1), :L)
ub = A\[c; f.(x[2:end])]

##adding the forward and midpoint solutions here as well for comparison
m = (x[1:end-1] + x[2:end])/2

uf = A \ [c; f.(x[1:end-1])]
um = A \ [c; f.(m)]

plot(x, sin.(x); label="sin(x)", legend=:bottomright)
scatter!(x, ub; label="backward")
scatter!(x, um; label="mid")
scatter!(x, uf; label="forward")
```



Comparing each method's errors, we see that the backward method has the same error as the forward method:

```
function indefint(x)
    h = step(x) # x[k+1]-x[k]
    n = length(x)
    L = Bidiagonal([1; fill(1/h, n-1)], fill(-1/h, n-1), :L)
end

function forward_err(u, c, f, n)
    x = range(0, 1; length = n)
    uᶠ = indefint(x) \ [c; f.(x[1:end-1])]
    norm(uᶠ - u.(x), Inf)
end

function mid_err(u, c, f, n)
    x = range(0, 1; length = n)
    m = (x[1:end-1] + x[2:end]) / 2 # midpoints
    uᵐ = indefint(x) \ [c; f.(m)]
    norm(uᵐ - u.(x), Inf)
end

function back_err(u, c, f, n)
    x = range(0,1;length=n)
    h=step(x)
    A = Bidiagonal([1; fill(1/h, n-1)], fill(-1/h, n-1), :L)
    ub = A\[c; f.(x[2:end])]
    norm(ub - u.(x), Inf)
end

c = 0 # u(0) = 0
f = x -> cos(x)
m = (x[1:end-1] + x[2:end])/2 # midpoints
ns = 10 .^ (1:8) # solve up to n = 10 million


scatter(ns, forward_err.(sin, 0, f, ns); xscale=:log10, yscale=:log10, label="forward")
scatter!(ns, mid_err.(sin, 0, f, ns); label="mid")
scatter!(ns, back_err.(sin, 0, f, ns); label="back",alpha=0.5)
plot!(ns, ns .^ (-1); label="1/n")
```

```
plot!(ns, ns .^ (-2); label="1/n^2")
```



Part two:

```
c = 0 # u(0) = 0
n = 10000

#functions defined in the solutions to problem sheet 2
f = x -> exp(exp(x)cos(x) + sin(x))
g = x -> prod([x] ./ (1:1000) .- 1)
function cont(n, x)
   ret = 2*one(x)
   for k = 1:n-1
       ret = 2 + (x-1)/ret
   end
   1 + (x-1)/ret
end

x = range(0,1;length=n)
h=step(x)
A = Bidiagonal([1; fill(1/h, n-1)], fill(-1/h, n-1), :L)
uf = A\[c; f.(x[2:end])]
ug = A\[c; g.(x[2:end])]
ucont = A\[c; cont.(1000, x[2:end])]

uf_int = uf[end]
ug_int = ug[end]
ucont_int = ucont[end]

println("first function: ")
println(uf_int)
println("second functions: ")
println(ug_int)
println("third function: ")
println(ucont_int)
```

Problem 2.2 (A) Implement indefinite-integration
where we take the average of the two grid points:

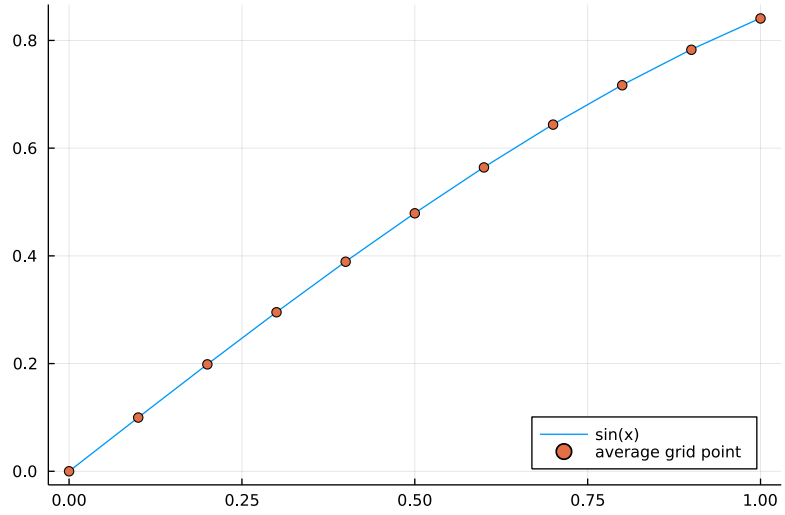
$$\frac{u'(x_{k+1}) + u'(x_k)}{2} \approx \frac{u_{k+1} - u_k}{h}$$

What is the observed rate-of-convergence using the ∞-norm for $f(x) = \cos x$ on the interval $[0, 1]$?
Does the method converge if the error is measured in the 1-norm?

SOLUTION

The implementation is as follows:

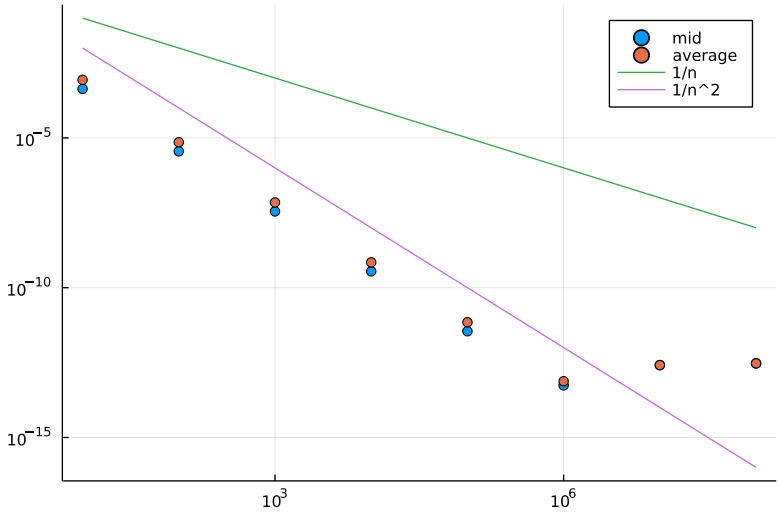
```
n = 10
x = range(0, 1; length=n+1)
h = 1/n
A = Bidiagonal([1; fill(1/h, n)], fill(-1/h, n), :L)
c = 0 # u(0) = 0
f = x -> cos(x)

f = f.(x) # evaluate f at all but last points
uₙ = A \ [c; (f[1:end-1] + f[2:end])/2]

plot(x, sin.(x); label="sin(x)", legend=:bottomright)
scatter!(x, uₙ; label="average grid point")

# print(norm(uₙ - sin.(x),Inf))
# norm(uₙ - sin.(x),1)
```



Comparing the error to the midpoint method, we see that the errors are very similar:

```
function average_err(u, c, f, n)
   x = range(0,1;length=n)
   h=step(x)
   A = Bidiagonal([1; fill(1/h, n-1)], fill(-1/h, n-1), :L)
   ua = A\[c; (f.(x[1:end-1]) + f.(x[2:end]))/2]
   norm(ua - u.(x), Inf)
end

c = 0 # u(0) = 0
f = x -> cos(x)
```

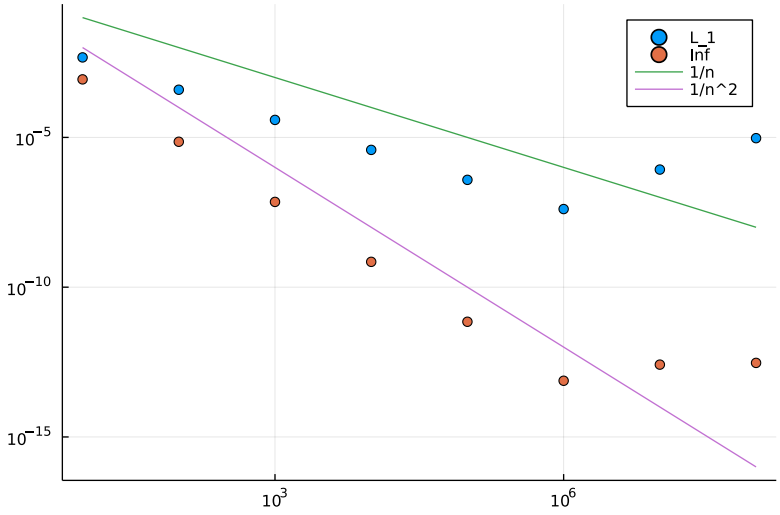
```
m = (x[1:end-1] + x[2:end])/2 # midpoints
ns = 10 .^ (1:8) # solve up to n = 10 million


scatter(ns, mid_err.(sin, 0, f, ns); xscale=:log10, yscale=:log10, label="mid")
scatter!(ns, average_err.(sin, 0, f, ns); label="average")
plot!(ns, ns .^ (-1); label="1/n")
plot!(ns, ns .^ (-2); label="1/n^2")
```




```
print(mid_err.(sin, 0, f, ns) - average_err.(sin, 0, f, ns))
```

Now looking at the L_1 norm, we see it is converging, but to a smaller error before it starts to increase:

```
function average_err_l1(u, c, f, n)
   x = range(0,1;length=n)
   h=step(x)
   A = Bidiagonal([1; fill(1/h, n-1)], fill(-1/h, n-1), :L)
   ua = A\[c; (f.(x[1:end-1]) + f.(x[2:end]))/2]
   norm(ua - u.(x), 1)
end

scatter(ns, average_err_l1.(sin, 0, f, ns); xscale=:log10, yscale=:log10, label="L_1")
scatter!(ns, average_err.(sin, 0, f, ns); label="Inf")
plot!(ns, ns .^ (-1); label="1/n")
plot!(ns, ns .^ (-2); label="1/n^2")
```



3. Euler methods

Problem 3.1 (B) Solve the following ODEs
using forward and/or backward Euler and increasing n, the number of time-steps,
until $u(1)$ is determined to 3 digits:

$$\begin{aligned} u(0) = 1, u'(t) = \cos(t)u(t) + t \\ v(0) = 1, v'(0) = 0, v''(t) = \cos(t)v(t) + t \\ w(0) = 1, w'(0) = 0, w''(t) = tw(t) + 2w(t)^2 \end{aligned}$$

If we increase the initial condition $w(0) = c > 1, w'(0)$
the solution may blow up in finite time. Find the smallest positive integer c
such that the numerical approximation suggests the equation
does not blow up.

SOLUTION

```
function first_eq(n)
   #this function takes n and returns the estimate of u(1) using n steps
   #define the range of t
   t = range(0, 1; length=n)
   #find the step-size h
   h = step(t)

   #preallocate memory
   u = zeros(n)
   #set initial condition
   u[1] = 1
   for k=1:n-1
      u[k+1] = (1+h*cos(t[k]))*u[k] + h*t[k]
   end
   u[end]
end
ns = 2 .^ (1:13)
println(first_eq.(ns)')
```

We see that $u(1) = 2.96$ to three digits.

```
#define A(t)
A = t -> [0 1; cos(t) 0]

function second_eq(n)
   #this function takes n and returns the estimate of v(1) using n steps
   #define the range of t
   t = range(0, 1; length=n)
   #find the step-size h
   h = step(t)

   #preallocate memory
   u = zeros(2,n)
   #set initial condition
   u[:,1] = [1.0, 0.0]
   for k=1:n-1
      u[:,k+1] = (I + h .* A(t[k]))*u[:,k] + h .* [0, t[k]]
   end
   u[1,end]
end
ns = 2 .^ (1:13)
println(second_eq.(ns)')
```

We see that $v(1)$ is 1.66 to three digits. Finally,

```
#define F(t)
function F(t, w)
  [w[2], t*w[1] + 2*w[1]*w[1]]
end

function third_eq(n=1000, c=1.0)
   #this function takes n and returns the estimate of w(1)
   #using n steps and with initial condition w(0) = c, with c defaulting to 0
   #if no argument is passed

   #define the range of t
   t = range(0, 1; length=n)
   #find the step-size h
   h = step(t)
   #preallocate memory
   w = zeros(2,n)
   #set initial condition
   w[:,1] = [c, 0.0]
   for k=1:n-1
      w[:,k+1] = w[:,k] + h .* F(t[k], w[:,k])
   end
   w[1,end]
end
ns = 2 .^ (1:18)
println(third_eq.(ns)')
```

For $c = 1$, we see that $w(1) = 2.80$ to 3 digits. Now consider for $c > 1$:

```
function third_eq(c)
   #this function takes n and returns the estimate of w(1)
   #using n steps and with initial condition w(0) = c, with c defaulting to 0
   #if no argument is passed
   n=100000
   #define the range of t
   t = range(0, 1; length=n)
   #find the step-size h
   h = step(t)
   #preallocate memory
   w = zeros(2,n)
   #set initial condition
   w[:,1] = [c, 0.0]
   for k=1:n-1
      w[:,k+1] = w[:,k] + h .* F(t[k], w[:,k])
   end
   w[1,end]
end
cs = 2:10
c_vals = third_eq.(cs)
```

It appears that $c = 2$ is the smallest positive integer greater than 1 for which the numerical approximation suggests the equation does not blow up.

Problem 3.2⋆ (B) For an evenly spaced grid $t_1, \dots, t_n$, use the approximation

$$\frac{u'(t_{k+1}) + u'(t_k)}{2} \approx \frac{u_{k+1} - u_k}{h}$$

to recast

$$\begin{aligned} u(0) &= c \\ u'(t) &= a(t)u(t) + f(t) \end{aligned}$$

as a lower bidiagonal linear system. Use forward-substitution to extend this to vector linear problems:

$$\begin{aligned} \mathbf{u}(0) &= \mathbf{c} \\ \mathbf{u}'(t) &= A(t)\mathbf{u}(t) + \mathbf{f}(t) \end{aligned}$$

SOLUTION

We have,
\begin{align}
\frac{u_{k+1} - u_k}{h} \approx \frac{u'(t_{k+1}) + u'(t_k)}{2} = \frac{a(t_{k+1})u_{k+1} + a(t_k)u_k}{2} + \frac{1}{2}(f(t_{k+1}) + f(t_k)),
\end{align}
so we can write,
\begin{align}
\left(\frac{1}{h} - \frac{a(t_{k+1})}{2}\right)u_{k+1} + \left(-\frac{1}{h} - \frac{a(t_k)}{2}\right)u_k = \frac{1}{2}(f(t_{k+1}) + f(t_k)).
\end{align}
With the initial condition $u(0) = c$, we can write the whole system as,

$$\begin{bmatrix} 1 & & & \\ -\frac{1}{h} - \frac{a(t_1)}{2} & \ddots & & \\ & \ddots & \frac{1}{h} - \frac{a(t_2)}{2} & \\ & & -\frac{1}{h} - \frac{a(t_{n-1})}{2} & \frac{1}{h} - \frac{a(t_n)}{2} \end{bmatrix} \mathbf{u} = \begin{bmatrix} c \\ \frac{1}{2}(f(t_1) + f(t_2)) \\ \vdots \\ \frac{1}{2}(f(t_{n-1}) + f(t_n)) \end{bmatrix},$$

which is lower bidiagonal.

Now if we wish to use forward substitution in a vector linear problem, we can derive in much the same way as above:

$$\left(\frac{1}{h}I - \frac{A(t_{k+1})}{2}\right)\mathbf{u}_{k+1} + \left(-\frac{1}{h}I - \frac{A(t_k)}{2}\right)\mathbf{u}_k = \frac{1}{2}(\mathbf{f}(t_{k+1}) + \mathbf{f}(t_k)),$$

to make the update equation,

$$\mathbf{u}_{k+1} = \left(I - \frac{h}{2}A(t_{k+1})\right)^{-1}\left(\left(I + \frac{h}{2}A(t_k)\right)\mathbf{u}_k + \frac{h}{2}(\mathbf{f}(t_{k+1}) + \mathbf{f}(t_k))\right),$$

with initial value,

$$\mathbf{u}_1 = \mathbf{c}.$$

Problem 3.3 (B) Implement the method designed in Problem 3.1 for the negative time Airy equation

$$u(0) = 1, u'(0) = 0, u''(t) = -tu(t)$$

on $[0, 50]$.
How many time-steps are needed to get convergence to 1% accuracy (the "eyeball norm")?

SOLUTION
We will work with,
$\mathbf{u}(t) = \begin{bmatrix} u(t) \\ u'(t) \end{bmatrix}$,
so that our differential equation is:

$$\mathbf{u}'(t) = \begin{bmatrix} u'(t) \\ u''(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -t & 0 \end{bmatrix}\mathbf{u}(t),$$

so that,

$$A(t) = \begin{bmatrix} 0 & 1 \\ -t & 0 \end{bmatrix},$$

and with initial conditions,

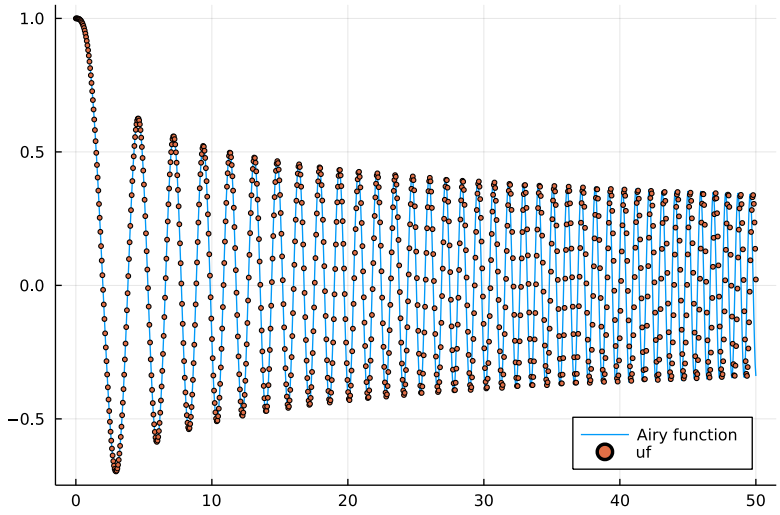
$$\mathbf{u}(0) = \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

We will use the method described in Problem 3.1, with $\mathbf{f}(t) = \mathbf{0}$:

$$\mathbf{u}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \mathbf{u}_{k+1} = \left(I - \frac{h}{2}A(t_{k+1})\right)^{-1}\left(I + \frac{h}{2}A(t_k)\right)\mathbf{u}_k.$$

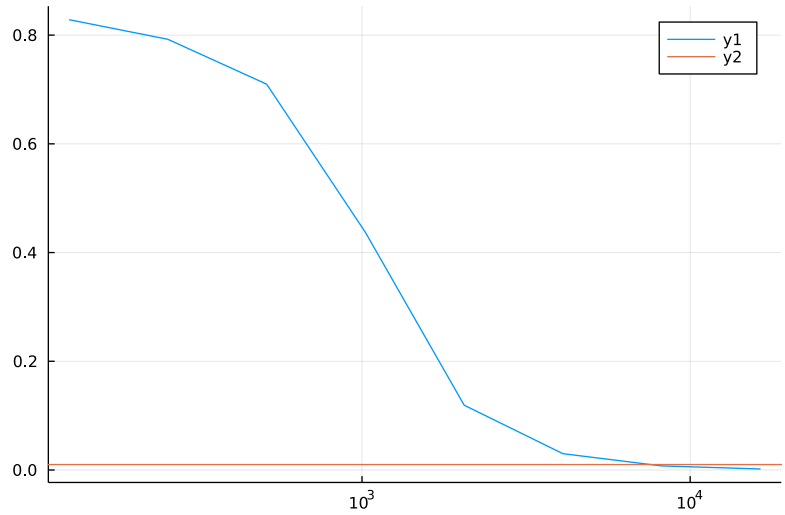
```
using SpecialFunctions
n = 1000
#define the range of t
t = range(0, 50; length=n)
```

```
#find the step-size h
h = step(t)
#define the function a
a = t -> [0 1; -t 0]

#initialise storage vector and set initial conditions
U=zeros(2, n)
U[:,1] = [1.0, 0.0]

#now iterate forward
for k = 1:n-1
   U[:,k+1] = (I - h/2 .* a(t[k+1])) \ ((I + h/2 .* a(t[k])) * U[:,k])
end

#solution found on wolfram alpha
u = t -> real(1/2 * 3^(1/6) * gamma(2/3) * (sqrt(3)airyai((-1 + 0im)^(1/3)*t) + airybi((-1+

plot(t, u.(t), label="Airy function")
scatter!(t, U[1,:], label="uf", legend=:bottomright, markersize=2)
```



To see when the error goes below 1%, consider the below:

```
n = 2 .^(7:14)
function relative_err(n)
   t = range(0, 50; length=n)
   #find the step-size h
   h = step(t)
   #initialise storage vector and set initial conditions
   U=zeros(2, n)
   U[:,1] = [1.0, 0.0]
   #now iterate forward
   for k = 1:n-1
       U[:,k+1] = (I - h/2 .* a(t[k+1])) \ ((I + h/2 .* a(t[k])) * U[:,k])
   end
   norm(U[1,:] - u.(t), Inf)/norm(u.(t), Inf)
end

plot(n, relative_err.(n), xscale=:log10)
plot!([0.01], seriestype=:hline)
```



Problem 3.4 (A) Implement Heat on a graph with $m = 50$ nodes with no forcing and initial condition $u_{m/2}(0) = 1$ and $u_k(0) = 0$, but where the first and last node are fixed to zero, that is replace the first and last rows of the differential equation with the conditions:

$$u_1(t) = u_m(t) = 0.$$

Find the time t such that $\|\mathbf{u}(t)\|_\infty < 10^{-3}$ to 2 digits.
Hint: Differentiate to recast the conditions as a differential equation.
Vary n, the number of time-steps used in Forward Euler, and increase T in the interval $[0, T]$ until the answer doesn't change.
Do a for loop over the time-slices to find the first that satisfies the condition.
(You may wish to call `println` to print the answer and `break` to exit the for loop).

SOLUTION

Following the hint, we will begin by writing a function called `heat_dissipation(T)`, which runs a simulation of the heat equation with the specified conditions up to time T. If the condition $\|\mathbf{u}(t)\|_\infty < 10^{-3}$ is met at a time $t^* < T$, then it will return t^*, else it will return T. We choose the value $n = 1000$ not too large so that we can run this on a large range of values for T. Also note that we use Backward Euler, which is more stable for smaller values of n; T can potentially be quite large, so Forward Euler may be unstable for even moderately large values of n.

```
function heat_dissipation(T)
    n=1000
    t = range(0, T; length=n)
    m=50
    #find the step-size h
    h = step(t)
    #define the matrix
    Δ = Tridiagonal([fill(1.0, m-2); 0], [0; fill(-2.0, m-2); 0], [0; fill(1.0, m-2)])

    #set initial conditions
    u = zeros(m,n)
    u[Int(m/2), 1] = 1
    for k = 1:n-1
        u[:,k+1] = (I - h*Δ)\u[:,k]
```

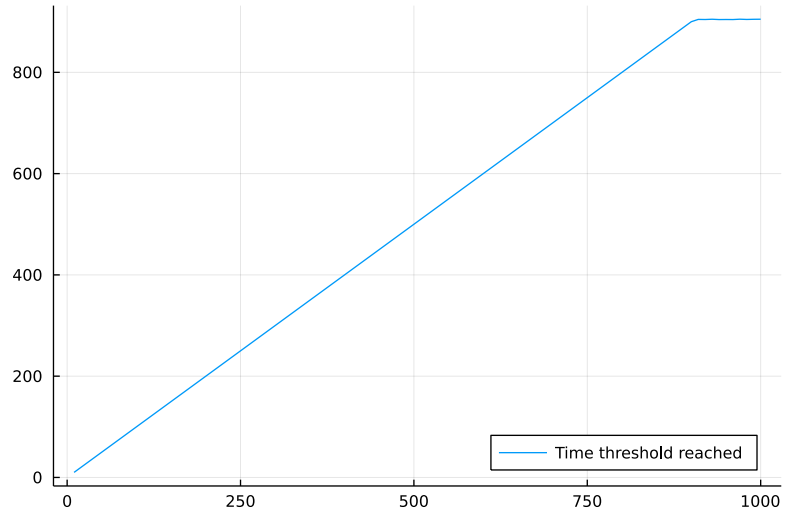
```
        u_inf = norm(u[:,k+1], Inf)
        if(u_inf < 0.001)
            return t[k+1]
        end
    end
    return t[n]
end
```

We run this on a large range of values for T. The function returns approximately constant (≈ 905) values when $T > 905$, so this suggests that our answer lies somewhere around 905.

```
Ts = 10:10:1000
ts = heat_dissipation.(Ts)
```

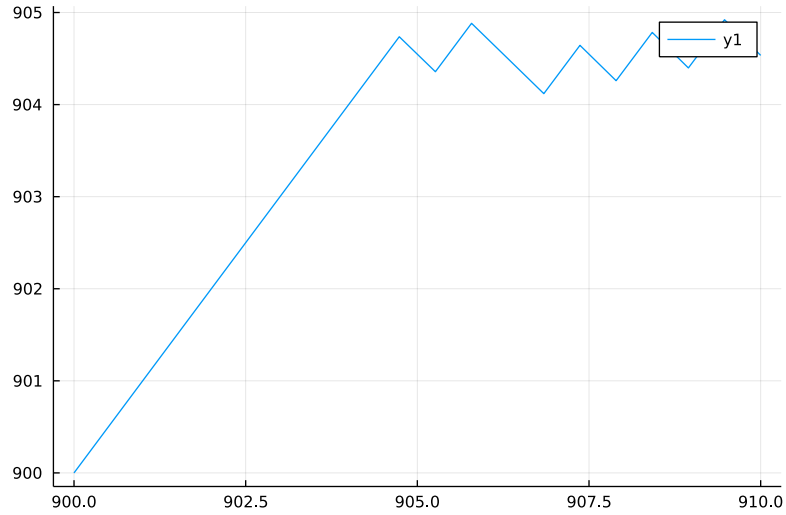
Plotting, we can clearly see that the time output by the function becomes the same towards the end of our range, so we will restrict our search to the end of this range.

```
plot(Ts, ts, label = "Time threshold reached", legend=:bottomright)
```



Zooming in:

```
Ts = range(900, 910,20)
ts = heat_dissipation.(Ts)
plot(Ts,ts)
```



This looks promising, but it seems like the time-output is somewhat unstable even after T is large enough. Inspecting the actual values of the output, we see that this is likely due to the step size we are using - it will be different for different values of T (as $h = \frac{T}{n}$), and so the smallest t in the discretise range may jump up and down if n is not large enough. To be sure of the answer to 2 decimal places, we will need n to be larger than $2\frac{T}{0.01} \approx 180000$. We will redefine our function with $n = 200000$, and run it on a few different values of T (that are definitely larger than our target time) to be sure we get the same answer to 2 decimal places.

```
function heat_dissipation_large_n(T)
    n=200000
    t = range(0, T; length=n)
    m=50
    #find the step-size h
    h = step(t)
    #define the matrix
    Δ = Tridiagonal([fill(1.0, m-2); 0], [0; fill(-2.0, m-2); 0], [0; fill(1.0, m-2)])

    #set initial conditions
    u = zeros(m,n)
    u[Int(m/2), 1] = 1
    for k = 1:n-1
        u[:,k+1] = (I - h*Δ)\u[:,k]
        u_inf = norm(u[:,k+1], Inf)
        if(u_inf < 0.001)
            return t[k+1]
        end
    end
    return t[n]
end

Ts = [903, 904, 905, 906, 907]
ts = heat_dissipation_large_n.(Ts)
```

We can see that each time we get 902.38 to 2 decimal places, so this is our answer.

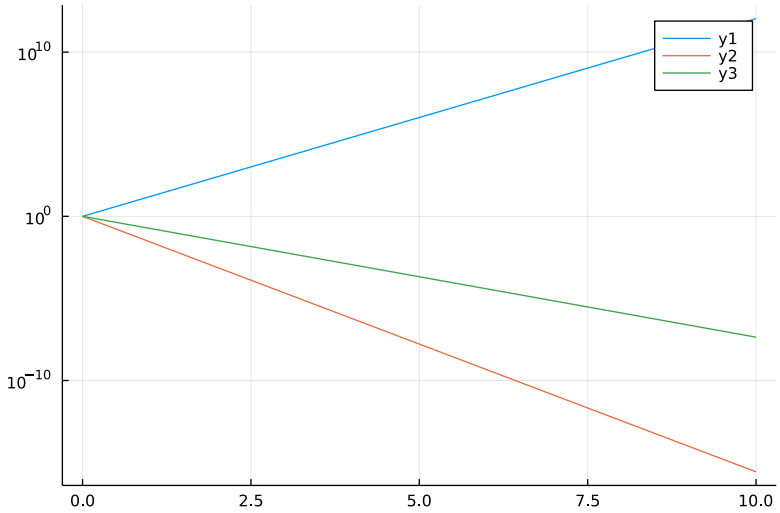
Problem 3.5 (B) Consider the equation

$$u(0) = 1, u'(t) = -10u(t)$$

What behaviour do you observe on $[0, 10]$ of forward, backward, and that of Problem 3.1 with a step-size of 0.5? What happens if you decrease the step-size to 0.01? (Hint: you may wish to do a plot and scale the y-axis logarithmically,)

SOLUTION

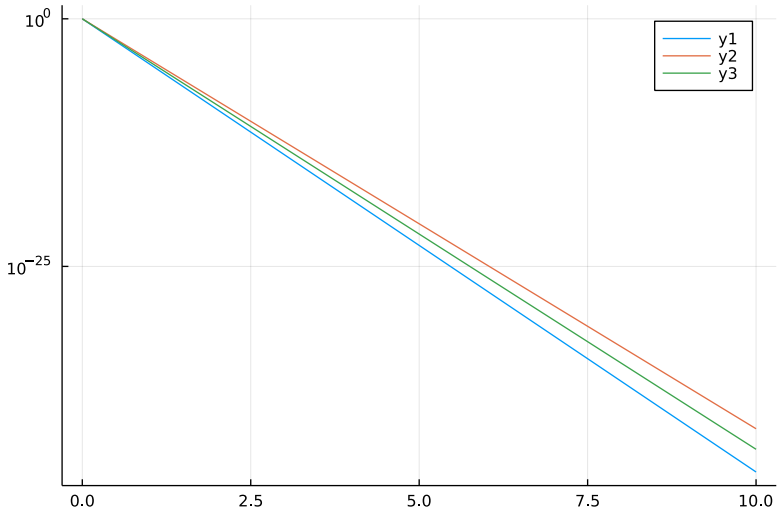
```
h = 0.5
t = range(0, 10; step=h)
n = length(t)
uᶠ = zeros(n)
uᵇ = zeros(n)
uᵗ = zeros(n)
uᶠ[1] = uᵇ[1] = uᵗ[1] = 1
a = -10
for k = 1:n-1
   uᶠ[k+1] = (1+h*a) * uᶠ[k]
   uᵇ[k+1] = (1-h*a) \ uᵇ[k]
   uᵗ[k+1] = (1-h*a/2) \ (1 + h*a/2) * uᵗ[k]
end

plot(t, abs.(uᶠ); yscale=:log10)
plot!(t, abs.(uᵇ); yscale=:log10)
plot!(t, abs.(uᵗ); yscale=:log10)
```



We observe that for the stepsize $h = 0.5$, the forward method blows up while the other methods appear to converge.

```
h = 0.01
t = range(0, 10; step=h)
n = length(t)
uᶠ = zeros(n)
uᵇ = zeros(n)
uᵗ = zeros(n)
uᶠ[1] = uᵇ[1] = uᵗ[1] = 1
for k = 1:n-1
   uᶠ[k+1] = (1+h*a) * uᶠ[k]
   uᵇ[k+1] = (1-h*a) \ uᵇ[k]
   uᵗ[k+1] = (1-h*a/2) \ (1 + h*a/2) * uᵗ[k]
```

```
end

nanabs(x) = iszero(x) ? NaN : abs(x)

plot(t, nanabs.(uᶠ); yscale=:log10)
plot!(t, nanabs.(uᵇ); yscale=:log10)
plot!(t, nanabs.(uᵗ); yscale=:log10)
```



For a smaller stepsize ($h = 0.01$), the forward method is also able to converge.

MATH50003 Numerical Analysis: Problem Sheet 7

This problem sheet explores condition numbers, indefinite integration,
and Euler's method.

Questions marked with a ⋆ are meant to be completed without using a computer.
Problems are denoted A/B/C to indicate their difficulty.

```
using LinearAlgebra, Plots, Test
```

1. Two-point boundary value problems

Problem 1.1 (C) Construct a finite-difference approximation to the
forced Helmholtz equation

$$\begin{aligned} u(0) &= 0 \\ u(1) &= 0 \\ u'' + k^2 u &= \mathrm{e}^x \end{aligned}$$

and find an n such the error is less than 10^{-4} when compared
with the true solution for $k = 10$:

$$u(x) = (-\cos(kx) + \mathrm{e}^x \cos(kx)^2 + \cot(k)\sin(kx) - \mathrm{e}\cos(k)\cot(k)\sin(kx) - \mathrm{e}\sin(k)\sin(kx) + \mathrm{e}^x \sin(kx$$

```
function helm(k, n)
   x = range(0, 1; length = n)
   h = step(x)
   # TODO: Create a SymTridiagonal discretisation
   T = SymTridiagonal(ones(n-2)*(-2/h^2 + k^2),ones(n-3)*1/h^2)
   u = T \ exp.(x[2:end-1])
  [0; u; 0]
end

k = 10
u = x -> (-cos(k*x) + exp(x)cos(k*x)^2 + cot(k)sin(k*x) - e*cos(k)cot(k)sin(k*x) - e*sin(k)

n = 2048 # TODO: choose n to get convergence
x = range(0, 1; length=n)
@test norm(helm(k, n) - u.(x)) ≤ 1E-4
```

Problem 1.2 (B) Discretisations can also be used to solve eigenvalue equations.
Consider the Schrödinger equation with quadratic oscillator:

$$u(-L) = u(L) = 0, -u'' + x^2 u = \lambda u$$

Approximate the eigenvalues using **`eigvals(A)`** (which returns the eigenvalues of a matrix **`A`**) with $L = 10$.
Can you conjecture their exact value if $L = \infty$? (Hint: they are integers and the eigenvalues closest to zero are most accurate.)

SOLUTION

```
L = 10
n = 1000
x = range(-L,L; length=n)
h = step(x)
eigvals(SymTridiagonal(fill(2/h^2,n-2)  + x[2:end-1].^2, fill(-1/h^2, n-3)))
```

On inspection of the smallest values, it seems that the positive odd integers are the eigenvalues for $L = \infty$. Increasing L (and also n) it becomes more obvious:

```
L = 100
n = 10000
x = range(-L,L; length = n)
h = step(x)
A = SymTridiagonal(x[2:end-1] .^ 2 .+ 2/h^2,ones(n-3)* (-1)/h^2)
sort((eigvals(A)))[1:20]
```

Problem 1.3⋆ (A) Consider Helmholtz with Neumann conditions:

$$u'(0) = c_0$$
$$u'(1) = c_1$$
$$u_{xx} + k^2 u = f(x)$$

Write down the finite difference approximation approximating
$u(x_k) \approx u_k$ on
an evenly spaced grid $x_k = (k-1)/(n-1)$ for $k = 1, \dots, n$
using the first order derivative approximation conditions:

$$u'(0) \approx (u_2 - u_1)/h = c_0$$
$$u'(1) \approx (u_n - u_{n-1})/h = c_1$$

Use pivoting to reduce the equation to one involving a
symmetric tridiagonal matrix.

SOLUTION

We have, with $u(x_k) = u_k$ (and using κ instead of k in the equation $u_{xx} + k^2 u = f(x)$ so as to avoid confusion with the indices):

$$\begin{aligned} \frac{u_2 - u_1}{h} &= c_0, \\ \frac{u_{k-1} - 2u_k + u_{k+1}}{h^2} + \kappa^2 u_k &= f(x_k), \quad \text{for } k = 2 : n-1 \\ \frac{u_n - u_{n-1}}{h} &= c_1, \end{aligned}$$

which we write in matrix form as:

$$\begin{bmatrix} -\frac{1}{h} & \frac{1}{h} & & & \\ \frac{1}{h^2} & \kappa^2 - \frac{2}{h^2} & \frac{1}{h^2} & & \\ & \ddots & \ddots & \ddots & \\ & & \frac{1}{h^2} & \kappa^2 - \frac{2}{h^2} & \frac{1}{h^2} \\ & & & -\frac{1}{h} & \frac{1}{h} \end{bmatrix} \mathbf{u} = \begin{bmatrix} c_0 \\ f(x_2) \\ \vdots \\ f(x_{n-1}) \\ c_1 \end{bmatrix},$$

which we can make symmetric tridiagonal by multiplying the first row by $1/h$ and the final row by $-1/h$:

$$\begin{bmatrix} -\frac{1}{h^2} & \frac{1}{h^2} & & & \\ \frac{1}{h^2} & \kappa^2 - \frac{2}{h^2} & \frac{1}{h^2} & & \\ & \ddots & \ddots & \ddots & \\ & & \frac{1}{h^2} & \kappa^2 - \frac{2}{h^2} & \frac{1}{h^2} \\ & & & \frac{1}{h^2} & -\frac{1}{h^2} \end{bmatrix} \mathbf{u} = \begin{bmatrix} \frac{c_0}{h} \\ f(x_2) \\ \vdots \\ f(x_{n-1}) \\ -\frac{c_1}{h} \end{bmatrix},$$

2. Convergence

Problem 2.1⋆ (B) For the equation

$$u(0) = c_0$$
$$u' + au = f(x)$$

where $a \in \mathbb{R}$ and $0 \le x \le 1$,
prove convergence as $n \to \infty$ for the method constructed in PS6 using the approximation
where we take the average of the two grid points:

$$\frac{u'(x_{k+1}) + u'(x_k)}{2} \approx \frac{u_{k+1} - u_k}{h}.$$

SOLUTION

Using the approximation from PS6 we obtain

$$\frac{(f(x_{k+1})+f(x_k))}{2} = \frac{(u'(x_{k+1})+u'(x_k))}{2} + \frac{a(u(x_{k+1})+u(x_k))}{2} \approx \frac{(u_{k+1}-u_k)}{h} + \frac{au_{k+1}}{2} + \frac{au_k}{2}$$

So we get

$$\left(\frac{a}{2} - \frac{1}{h}\right)u_k + \left(\frac{a}{2} + \frac{1}{h}\right)u_{k+1} = \frac{f(x_{k+1})+f(x_k)}{2}$$

We want to prove that $\sup_{k=1,\dots,n-1} |u(x_k) - u_k|$ converges to 0 as $n \to \infty$.

Take $\hat{u} = [u_0, \dots, u_{n-1}]^T$ and rewrite the system as

$$\hat{L}\hat{u} = \begin{bmatrix} c_0 \\ \hat{f}^f \end{bmatrix}$$

where $f_k = \frac{f(x_k)+f(x_{k-1})}{2}$, $k = 1, \dots, n-1$ and

$$\hat{L} = \begin{bmatrix} 1 & & & & \\ \frac{a}{2} - \frac{1}{h} & \frac{a}{2} + \frac{1}{h} & & & \\ & \frac{a}{2} - \frac{1}{h} & \frac{a}{2} + \frac{1}{h} & & \\ & & \ddots & \ddots & \\ & & & \frac{a}{2} - \frac{1}{h} & \frac{a}{2} + \frac{1}{h} \end{bmatrix}$$

Note that $\hat{L}$ is lower bidiagonal.

Now, similarly to Euler's methods convergence theorem, we study consistency and stability.

Consistency:

Our discretisation approximates the true equation.

$$\hat{L}u = \begin{bmatrix} c_0 \\ \frac{u(x_1)-u(x_0)}{h} + \frac{a}{2}(u(x_1) + u(x_0)) \\ \vdots \\ \frac{u(x_{n-1})-u(x_{n-2})}{h} + \frac{a}{2}(u(x_{n-1}) + u(x_{n-2})) \end{bmatrix} = \begin{bmatrix} c_0 \\ \frac{1}{2}\left(\frac{u(x_1)-u(x_0)}{h} + \frac{u(x_1)-u(x_0)}{h} + a(u(x_1) + u(x_0))\right) \\ \vdots \\ \frac{1}{2}\left(\frac{u(x_{n-1})-u(x_{n-2})}{h} + \frac{u(x_1)-u(x_0)}{h} + a(u(x_{n-1}) + u(x_{n-2}))\right) \end{bmatrix}$$

where $x_k \le \tau_k, \sigma_k \le x_{k+1}$, and uniform boundedness implies that $\|\delta\|_\infty = O(h)$

Stability:

The inverse does not blow up the error.

$$\hat{L} = \underbrace{\begin{bmatrix} 1 & & & \\ & \left(\frac{a}{2} + \frac{1}{h}\right)^{-1} & & \\ & & \ddots & \\ & & & \left(\frac{a}{2} + \frac{1}{h}\right)^{-1} \end{bmatrix}}_{D} \underbrace{\begin{bmatrix} 1 & & & \\ \left(\frac{a}{2} + \frac{1}{h}\right)\left(\frac{a}{2} - \frac{1}{h}\right) & 1 & & \\ & \ddots & \ddots & \\ & & \left(\frac{a}{2} + \frac{1}{h}\right)\left(\frac{a}{2} - \frac{1}{h}\right) & 1 \end{bmatrix}}_{L}$$

Thus, we have $\|L^{-1}\|_{1\to\infty} \le \left|\frac{a^2}{4} - \frac{1}{h^2}\right|^{-(n-1)} \le \left|\frac{a^2}{4} - n^2\right|^{-(n-1)} \left|\frac{4}{a^2+4n^2}\right|^{n-1} = O(1)$

Note that $\left|\frac{a}{2} + \frac{1}{h}\right|^{-1} = \left|\frac{h}{\frac{ah}{2}+1}\right| \leq 2h$, as $h \to 0$ (taking h small enough)

Combining stability and consistency we have

$$\|\mathbf{u}^f - \mathbf{u}\|_\infty = \|\hat{L}^{-1}(\hat{L}\mathbf{u}^f - \hat{L}\mathbf{u})\|_\infty = \|L^{-1}D^{-1}\begin{bmatrix}0\\ \delta\end{bmatrix}\|_\infty \leq 2h\|L^{-1}\|_{1\to\infty}\|\delta\|_1 = O(h)$$

Problem 2.2⋆ (A) Consider the matrices

$$L = \begin{bmatrix} 1 & & & & \\ -a_1 & 1 & & & \\ & -a_2 & 1 & & \\ & & \ddots & \ddots & \\ & & & -a_{n-1} & 1 \end{bmatrix}, \qquad T = \begin{bmatrix} 1 & & & & \\ -a & 1 & & & \\ & -a & 1 & & \\ & & \ddots & \ddots & \\ & & & -a & 1 \end{bmatrix}.$$

By writing down the inverse explicitly prove that if $|a_k| \leq a$ then

$$\|L^{-1}\|_{1\to\infty} \leq \|T^{-1}\|_{1\to\infty}.$$

Use this to prove convergence as $n \to \infty$ of forward Euler for

$$\begin{aligned} u(0) &= c_0 \\ u'(x) - a(x)u(x) &= f(x) \end{aligned}$$

SOLUTION

Since

$$L^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \dots & 0 \\ a_1 & 1 & 0 & 0 & 0 & \dots & 0 \\ a_1a_2 & a_2 & 1 & 0 & 0 & \dots & 0 \\ a_1a_2a_3 & a_2a_3 & a_3 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \vdots & & \ddots & \ddots & 1 & 0 \\ \prod_{i=1}^{n-1} a_i & \prod_{i=2}^{n-1} a_i & \dots & \dots & a_{n-2}a_{n-1} & a_{n-1} & 1 \end{bmatrix}$$

and

$$T^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \dots & 0 \\ a & 1 & 0 & 0 & 0 & \dots & 0 \\ a^2 & a & 1 & 0 & 0 & \dots & 0 \\ a^3 & a^2 & a & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \vdots & & \ddots & \ddots & 1 & 0 \\ a^{n-1} & a^{n-2} & \dots & \dots & a^2 & a & 1 \end{bmatrix}$$

Then, $\forall x$

$$\|T^{-1}x\|_\infty = \max_i |(T^{-1}x)_i| = \max_i \left| x_i + \sum_{j=1}^{i-1} a^{i-j}x_j \right| = \begin{cases} 1 & if\ a \in [0,1] \\ a^{n-1} & if\ a \geq 1 \end{cases}$$

since, given $b = \max\{1, a\}$,

$\max_i \left|x_i + \sum_{j=1}^{i-1} a^{i-j}x_j\right| \leq \max_i \left(|x_i| + \sum_{j=1}^{i-1} \left|a^{i-j}x_j\right|\right) \leq b^n \sum_{j=1}^{n} |x_j| = b^n\|x\|_1$

thus,

$\|T^{-1}\|_{1\to\infty} = \sup_{x\neq 0} \frac{\|T^{-1}x\|_\infty}{\|x\|_1} \leq b^n$ and, in particular,
$\|T^{-1}\|_{1\to\infty} = b^n$

since $\frac{|T^{-1}x|_\infty}{\|x\|_1} = b^n$ it is obtained using
$x = \begin{cases} e_1 & b = 1 \\ e_n & b = a \end{cases}$

Moreover, $|a_j| \leq b, \forall j = 1, \dots, n$, thus,

$\|L^{-1}x\|_\infty = \max_i |(L^{-1}x)_i| = \max_i \left|x_i + \sum_{j=1}^{i-1} a_j \dots a_{i-1} x_j\right| \leq \max_i |x_i| + \sum_{j=1}^{i-1} |a_j \dots a_{i-1} x_j| \leq b^n \|x\|_1$

Hence,

$\|L^{-1}\|_{1\to\infty} = \sup_x \frac{\|L^{-1}x\|_\infty}{\|x\|_1} \leq b^n = \|T^{-1}\|_{1\to\infty}$

Now we prove convergence for the forward Euler method as $n \to \infty$ for

$$\begin{aligned} u(0) &= c_0 \\ u'(x) &= a(x)u(x) + f(x) \end{aligned}$$

Using equidistanced (with step h) points $x_0, \dots, x_{n-1}$, we use the approximations $u(x_k) \approx u_k$, where
$u_0 = c_0$ and
$u_{k+1} = u_k + h\left(a(x_k)u_k + f(x_k)\right)$

In order to study convergence we consider the limit as $n \to \infty$ of
$\sup_{i=1,\dots,n-1} |u_i - u(x_i)|$

Similarly to Euler's methods convergence theorem, we study consistency and stability.

In order to apply the theorem we note that we can define $a_k = a(x_k)$, $k = 1, \dots n-1$ and we have that for every k, $|a_k| \leq a := max_{i=1,n-1}|a_i|$.

Consistency:

Our discretisation approximates the true equation.

$$\hat{L}u = \begin{bmatrix} c_0 \\ \frac{u(x_1)-u(x_0)}{h} - a_1 u(x_0) \\ \vdots \\ \frac{u(x_{n-1})-u(x_{n-2})}{h} - a_{n-1}u(x_{n-2}) \end{bmatrix} = \begin{bmatrix} c_0 \\ u'(x_0) - a_1 u(x_0) + u''(\tau_0)h \\ \vdots \\ u'(x_{n-2}) - a_{n-1}u(x_{n-2}) + u''(\tau_{n-2})h \end{bmatrix} = \begin{bmatrix} c_0 \\ f(x_0) + u''(\tau_0)h \\ \vdots \\ f(x_{n-2}) + u''(\tau_{n-2})h \end{bmatrix}$$

where $x_k \leq \tau_k \leq x_{k+1}$, and uniform boundedness implies that $\|\delta\|_\infty = O(h)$

Stability:

The inverse does not blow up the error.
First write, for $l_k = 1 - a_k$

$$\hat{L} = \underbrace{\begin{bmatrix} 1 & & & \\ & h^{-1} & & \\ & & \ddots & \\ & & & h^{-1} \end{bmatrix}}_{D} \underbrace{\begin{bmatrix} 1 & & & \\ -l_1 & 1 & & \\ & \ddots & \ddots & \\ & & -l_{n-1} & 1 \end{bmatrix}}_{L}$$

Thus, we have $\|L^{-1}\|_{1\to\infty} \le \|T^{-1}\|_{1\to\infty} = O(1)$

Combining stability and consistency we have

$$\|\mathbf{u}^f - \mathbf{u}\|_\infty = \|\hat{L}^{-1}(\hat{L}\mathbf{u}^f - \hat{L}\mathbf{u})\|_\infty = \|L^{-1}D^{-1}\begin{bmatrix}0\\ \delta\end{bmatrix}\|_\infty \le h\|L^{-1}\|_{1\to\infty}\|\delta\|_1 = O(h)$$

3. Fourier series

Problem 3.1⋆ (C) Give explicit formulae for f_k and f_k^n for the following functions:

$$\cos\theta, \cos 4\theta, \sin^4\theta, \frac{3}{3-\mathrm{e}^{\mathrm{i}\theta}}, \frac{1}{1-2\mathrm{e}^{\mathrm{i}\theta}}$$

Hint: You may wish to try the change of variables $z = \mathrm{e}^{-\mathrm{i}\theta}$.

SOLUTION

The explicit formulae for f_k can be found by direct computation. Making use of trigonometric identities and noting that cos x and sin x are even and odd respectively on a periodic interval, hence orthogonal

1. $f_k = \frac{1}{2\pi}\int_0^{2\pi} \cos\theta \exp^{-ik\theta}\, d\theta = \frac{1}{2\pi}\int_0^{2\pi} (\cos\theta\cos k\theta) d\theta$

   For $k = 1$, yields

   $$f_k = \frac{1}{2\pi}\int_0^{2\pi} \cos^2\theta d\theta = \frac{1}{2}$$

   and $k \neq 1$ using standard trigonometric identity

   $$f_k = \frac{1}{4\pi}\int_0^{2\pi} \cos\theta(1-k) + \cos\theta(1+k) d\theta = 0$$

2. similar to 1), replace cases $k =, \neq 1$ by $k =, \neq 4$

3. Either use Euler's formula to expand $\sin^4\theta$ into exponentials or use the identity $sin^4\theta = \frac{3-4\cos 2\theta+\cos 4\theta}{8}$

   Noting that $\sin^4\theta$ is an even function yields $f_k = \frac{1}{2\pi}\int_0^{2\pi} \sin^4\theta \exp^{-ik\theta}\, d\theta = \frac{1}{2\pi}\int_0^{2\pi} (\sin^4\theta\cos k\theta) d\theta$
   Using the above identity we need to take care of the case $k \neq 2, 4$, which is zero as we have seen in 1).
   For special cases $k = 2, 4$ we have $f_2 = -\frac{1}{4}$ and $f_4 = \frac{1}{16}$.

4. We will have to separate the cases $k < 0$ and $k \ge 0$.

($k \ge 0$) Make the substition $z = 3\mathrm{e}^{-\mathrm{i}\theta}$ and integrating over the contour of the circle of radius 3
$f_k = \frac{-i}{2\pi}\frac{1}{3^{k+1}}\int_{|z|=3} \frac{z^k}{1-z} dz$
then by the residue theorem yields $f_k = \frac{1}{3^k}$.

($k < 0$) Make the substition $z = \mathrm{e}^{\mathrm{i}\theta}$ and observe that the pole lies outside of the contour of radius 1, hence $f_k = 0$ for all $k < 0$.

5. We will have to separate the cases $k < 0$ and $k \ge 0$.

($k \ge 0$) Make the substition $z = \frac{\mathrm{e}^{-i\theta}}{2}$ and observe that the pole lies outside of the contour of radius $\frac{1}{2}$, hence $f_k = 0$ for all $k \ge 0$.

($k < 0$) Make the substition $z = 2\mathrm{e}^{\mathrm{i}\theta}$ and integrating over the contour of the circle of radius 2
$\mathfrak{f}_k = \frac{i}{2\pi}\frac{1}{2^{k+1}}\int_{|z|=2}\frac{z^{k-1}}{z-1}dz$
then by the residue theorem yields $\mathfrak{f}_k = \frac{-1}{2^{|k|}}$.

For the student unfamiliar with complex analysis:

4*) Substitute for $z = \frac{\mathrm{e}^{-\mathrm{i}\theta}}{3}$ such that $f(\theta) = \frac{1}{1-z}$
then write the function as a geometric series $\Sigma_j z^j$ plug in

$\mathfrak{f}_k = \frac{1}{2\pi 3^j}\int_0^{2\pi}\Sigma_j \mathrm{e}^{i\theta(j-k)}d\theta$

and finally use the Lemma "Discrete orthogonality" from the lecture notes to show the result above.

5*) Similar idea to 4) however notice that this time we only have non-zero contributions for $k < 0$.

Problem 3.2⋆ (B) Prove that if the first $\lambda - 1$ derivatives $f(\theta), f'(\theta), \ldots, f^{(\lambda-1)}(\theta)$
are 2π-periodic and $f^{(\lambda)}$ is uniformly bounded that

$$|\mathfrak{f}_k| = O(|k|^{-\lambda}) \qquad \text{as } |k| \to \infty$$

Use this to show for the Taylor case ($0 = \mathfrak{f}_{-1} = \mathfrak{f}_{-2} = \cdots$) that

$$|f(\theta) - \sum_{k=0}^{n-1}\mathfrak{f}_k \mathrm{e}^{\mathrm{i}k\theta}| = O(n^{1-\lambda})$$

SOLUTION
A straightforward application of integration by parts yields the result

$\mathfrak{f}_k = \frac{1}{2\pi}\int_0^{2\pi} f(\theta)\mathrm{e}^{-ik\theta}d\theta = \frac{(-i)^\lambda}{2\pi k^\lambda}\int_0^{2\pi} f^{(\lambda)}(\theta)\mathrm{e}^{-ik\theta}d\theta$
given that $f^{(\lambda)}$ is uniformly bounded, the second part follows directly from this result

$$|\sum_{k=n}^{\infty}\mathfrak{f}_k \mathrm{e}^{\mathrm{i}k\theta}| \le \sum_{k=n}^{\infty}|\mathfrak{f}_k| \le C\sum_{k=n}^{\infty}k^{-\lambda}$$

for some constant C.

Problem 3.3⋆ (C)
If f is a trigonometric polynomial ($\mathfrak{f}_k = 0$ for $|k| > m$) show
for $n \ge 2m+1$ we can exactly recover f:

$$f(\theta) = \sum_{k=-m}^{m}\mathfrak{f}_k^n \mathrm{e}^{\mathrm{i}k\theta}$$

SOLUTION
This proof is nearly identical to the proof of "Theorem (Taylor series converges)" in the lecture notes. Only now one has to also subtract the negative coefficients from the negative approximate coefficients in the chain of arguments.

Problem 3.4⋆ (B) For the general (non-Taylor) case and $n = 2m+1$, prove convergence for

$$f_{-m:m}(\theta) := \sum_{k=-m}^{m}\mathfrak{f}_k^n \mathrm{e}^{\mathrm{i}k\theta}$$

to $f(\theta)$ as $n \to \infty$.
What is the rate of convergence if the first $\lambda - 1$ derivatives $f(\theta), f'(\theta), \ldots, f^{(\lambda-1)}(\theta)$
are 2π-periodic and $f^{(\lambda)}$ is uniformly bounded?

SOLUTION

Observe that by aliasing (see corollary in lecture notes) and triangle inequality we have the following

$|f_k^n - f_k| \leq \sum_{p=1}^{\infty}(|f_{k+pm}| + |f_{k-pm}|)$

Using the result from Problem 3.2 yields

$|f_k^n - f_k| \leq \frac{C}{n^\lambda}\sum_{p=1}^{\infty}\frac{1}{(p+\frac{k}{n})^\lambda} + \frac{1}{(p-\frac{k}{n})^\lambda}$

now we pick $|q| < \frac{1}{2}$ (such that the estimate below will hold for both summands above) and construct an integral with convex and monotonocally decreasing integrand such that

$(p+q)^{-\lambda} < \int_{p-\frac{1}{2}}^{p+\frac{1}{2}}(x+q)^{-\lambda}dx$

more over summing over the left-hand side from 1 to ∞ yields a bound by the integral:

$\int_{\frac{1}{2}}^{\infty}(x+q)^{-\lambda}dx = \frac{1}{\lambda}(\frac{1}{2}+q)^{-\lambda+1}$

Finally let $q = \pm\frac{k}{n}$ to achieve the rate of convergence

$|f_k^n - f_k| \leq \frac{C_\lambda}{n^\lambda}\left((\frac{1}{2}+k/n)^{-\lambda+1} + ((\frac{1}{2}-k/n))^{-\lambda+1}\right)$

where C_λ is a constant depending on λ. Note that it is indeed important to split the n coefficients equally over the negative and positive coefficients as stated in the notes, due to the estatime we used above.