[09-Sep-2021] Homework 1

Please work the following problems from our text:

- Problems 2.4, 2.5, 2.9, 2.11
- Problems 3.1, 3.3, 3.11, 3.12

2.4 Suppose $f : \mathbb{R} \to \mathbb{R}$ is infinitely differentiable, and we wish to write algorithms for finding x^* minimizing $f(x)$. Our algorithm outputs x_{est}, an approximation of x^*. Assuming that in our context this problem is equivalent to finding roots of $f'(x)$, write expressions for:

(a) Forward error of the approximation

(b) Backward error of the approximation

(c) Conditioning of this minimization problem near x^*

This is the same as solving $g(x)=0$ where $g(x)=f'(x)$ and the analysis is the same as Example 2.10. Setting $x^* = x+\varepsilon$ and replacing f by g yields

Example 2.10 (Root-finding)**.** Suppose that we are given a smooth function $g : \mathbb{R} \to \mathbb{R}$ and want to find roots x with $g(x) = 0$. By Taylor's theorem, $g(x + \varepsilon) \approx g(x) + \varepsilon g'(x)$ when $|\varepsilon|$ is small. Thus, an approximation of the condition number for finding the root x is given by

$$\frac{\text{forward error}}{\text{backward error}} = \frac{|(x+\varepsilon) - x|}{|g(x+\varepsilon) - g(x)|} \approx \frac{|\varepsilon|}{|\varepsilon g'(x)|} = \frac{1}{|g'(x)|} = \frac{1}{|f''(x)|}$$

(a) (b) (c)

This approximation generalizes the one in Example 2.9. If we do not know x, we cannot evaluate $g'(x)$, but if we can examine the form of g and *bound* $|g'|$ near x, we have an idea of the worst-case situation.

2.5 Suppose we are given a list of floating-point values $x_1, x_2, \ldots, x_n$. The following quantity, known as their "log-sum-exp," appears in many machine learning algorithms:

$$\ell(x_1, \ldots, x_n) \equiv \ln \left[\sum_{k=1}^{n} e^{x_k} \right].$$

(a) The value $p_k \equiv e^{x_k}$ often represents a probability $p_k \in (0, 1]$. In this case, what is the range of possible x_k's?

(b) Suppose many of the x_k's are very negative ($x_k \ll 0$). Explain why evaluating the log-sum-exp formula as written above may cause numerical error in this case.

(c) Show that for any $a \in \mathbb{R}$,

$$\ell(x_1, \ldots, x_n) = a + \ln \left[\sum_{k=1}^{n} e^{x_k - a} \right].$$

To avoid the issues you explained in 2.5b, suggest a value of a that may improve the stability of computing $\ell(x_1, \ldots, x_n)$.

(a) $\{x : e^x \in (0,1]\} \simeq \{\log p : p \in (0,1]\} = (-\infty, 0)$

(b) If $x_k \ll 0$ for many values of k, then some of the terms in the sum are very small compared to the others. When a small number is added to a large number, the small number looses precision.

In the present case evaluating $\ln(\alpha+\beta)$ where α is large compared to β is best done as

$$\ln(\alpha+\beta) \simeq \ln\left(\alpha\left(1+\frac{\beta}{\alpha}\right)\right) = \ln\alpha + \ln\left(1+\frac{\beta}{\alpha}\right) = \ln\alpha + \operatorname{log1p}\left(\frac{\beta}{\alpha}\right)$$

where $\operatorname{log1p}(\varepsilon) \simeq \ln(1+\varepsilon)$ is evaluated using the built-in function obtained from the Taylor series

$$\ln(1+\varepsilon) \simeq \varepsilon - \frac{1}{2}\varepsilon^2 + \frac{1}{3}\varepsilon^3 - \frac{1}{4}\varepsilon^4 + \cdots \qquad \text{for } \varepsilon \text{ very small.}$$

(c) $\ln\left[\sum_{k=1}^{n} e^{x_k}\right]$ may be evaluated more accurately by dividing the sum into two sums: a sum of very small terms plus a sum of larger terms. In particular, let $\mathcal{S} = \{k : x_k \ll 0\}$ and $\mathcal{L} = \{1, \ldots, n\} \setminus \mathcal{S}$. Then

$$\sum_{k=1}^{n} e^{x_k} \simeq \alpha + \beta \qquad \text{where } \alpha = \sum_{k \in \mathcal{L}} e^{x_k} \text{ and } \beta = \sum_{k \in \mathcal{S}} e^{x_k}$$

If follows, setting $a = \ln \alpha$ that $\alpha = e^a$

$$\ln \sum_{k=1}^{n} e^{x_k} \simeq \ln \alpha + \ln\left(1 + \frac{\beta}{\alpha}\right) \simeq a + \ln\left(1 + \beta e^{-a}\right)$$

$$\simeq a + \ln\left(\left(\alpha \sum_{k=1}^{n} + \beta\right) e^{-a}\right) = a + \ln\left(\sum_{k=1}^{n} e^{x_k - a}\right)$$

Note, in this case it is still important to evaluate $\ln\left(\sum_{k=1}^{n} e^{x_k - a}\right)$ using the log1p function as $\text{log1p}\left(\frac{\beta}{\alpha}\right)$ as suggested earlier.

2.9 In this problem, we continue to explore the conditioning of root-finding. Suppose $f(x)$ and $p(x)$ are smooth functions of $x \in \mathbb{R}$.

(a) Thanks to inaccuracies in how we evaluate or express $f(x)$, we might accidentally compute roots of a perturbation $f(x) + \varepsilon p(x)$. Take x^* to be a root of f, so $f(x^*) = 0$. If $f'(x^*) \neq 0$, for small ε we can write a function $x(\varepsilon)$ such that $f(x(\varepsilon)) + \varepsilon p(x(\varepsilon)) = 0$, with $x(0) = x^*$. Assuming such a function exists and is differentiable, show:

$$\left.\frac{dx}{d\varepsilon}\right|_{\varepsilon=0} = -\frac{p(x^*)}{f'(x^*)}.$$

(b) Assume $f(x)$ is given by Wilkinson's polynomial [131]:

$$f(x) \equiv (x-1)\cdot(x-2)\cdot(x-3)\cdot\cdots\cdot(x-20).$$

We could have expanded $f(x)$ in the monomial basis as $f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{20}x^{20}$, for appropriate choices of $a_0, \ldots, a_{20}$. If we express the coefficient a_{19} inaccurately, we could use the model from Exercise 2.9a with $p(x) \equiv x^{19}$ to predict how much root-finding will suffer. For these choices of $f(x)$ and $p(x)$, show:

$$\left.\frac{dx}{d\varepsilon}\right|_{\varepsilon=0, x^*=j} = -\prod_{k\neq j}\frac{j}{j-k}.$$

(c) Compare $\frac{dx}{d\varepsilon}$ from the previous part for $x^* = 1$ and $x^* = 20$. Which root is more stable to this perturbation?

(a) Use implicit differentiation

$$\frac{d}{d\varepsilon}\left(f(x(\varepsilon)) + \varepsilon p(x(\varepsilon)) = 0,\right)$$

Thus $f'(x(\varepsilon))\,x'(\varepsilon) + p(x(\varepsilon)) + \varepsilon p'(x(\varepsilon))\,x'(\varepsilon) = 0$

Set $\varepsilon = 0$ to obtain:

$$f'(x^*)\,x'(0) + p(x^*) = 0$$

Therefore $\left.\frac{dx_\varepsilon}{d\varepsilon}\right|_{\varepsilon=0} = x'(0) = \frac{-p(x^*)}{f'(x^*)}$

(b) Set $f(x) \equiv (x-1)\cdot(x-2)\cdot(x-3)\cdot\cdots\cdot(x-20).$

$p(x) \equiv x^{19}$

Then for $x^* = j$ we have

$$\left.\frac{dx_\varepsilon}{d\varepsilon}\right|_{\varepsilon=0} = \frac{-p(j)}{f'(j)}$$

Differentiating,

$$f'(x) = \frac{d}{dx}\prod_{k=1}^{n}(x-k) = \sum_{l=1}^{k}\prod_{k\neq l}(x-k)$$

So

$$f'(j) = \prod_{k\neq j}(j-k)$$

It follows that

$$\left.\frac{dx_\varepsilon}{d\varepsilon}\right|_{\varepsilon=0} = \frac{-P(j)}{f'(j)} = \frac{-j^{19}}{\prod_{k\neq j}(j-k)} = -\prod_{k\neq j}\left(\frac{j}{j-k}\right)$$

(c) When $j=1$ we obtain $\left.\frac{dx_\varepsilon}{d\varepsilon}\right|_{\varepsilon=0} = -\prod_{k\neq 1}\left(\frac{1}{1-k}\right) \approx 3.5477$

```
julia> ks=[1:19;]
19-element Vector{Int64}:
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
 16
 17
 18
 19

julia> sum(20 ./ (20 .- ks))
70.95479314287364
```

```
julia> ks=[2:20;]
19-element Vector{Int64}:
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
 16
 17
 18
 19
 20

julia> sum(1 ./ (1 .- ks))
-3.5477396571436826
```

When $j=20$ we obtain $\left.\frac{dx_\varepsilon}{d\varepsilon}\right|_{\varepsilon=0} = -\prod_{k\neq 20}\left(\frac{20}{20-k}\right) \approx 70.955$

It follows that the root at $j=1$ is more stable.

2.11 One technique for tracking uncertainty in a calculation is the use of *interval arithmetic*. In this system, an uncertain value for a variable x is represented as the interval $[x] \equiv [\underline{x}, \overline{x}]$ representing the range of possible values for x, from $\underline{x}$ to $\overline{x}$. Assuming infinite-precision arithmetic, give update rules for the following in terms of $\underline{x}$, $\overline{x}$, $\underline{y}$, and $\overline{y}$:

- $[x] + [y]$
- $[x] - [y]$
- $[x] \times [y]$
- $[x] \div [y]$
- $[x]^{1/2}$

Additionally, propose a conservative modification for finite-precision arithmetic.

- $[x]+[y] = [\underline{x}+\underline{y}, \overline{x}+\overline{y}]$

  lower two bounds is a lower bound

  upper bounds is a upper bound.

- $[x]-[y] = [\underline{x}, \overline{x}] - [\underline{y}, \overline{y}] = [\underline{x}, \overline{x}] + [-\overline{y}, -\underline{y}] = [\underline{x}-\overline{y}, \overline{x}-\underline{y}]$

  mult by -1 changes order of inequalities

- $[x] \times [y]$ There are lots of cases depending on whether $\underline{x}$, $\overline{x}$, $\underline{y}$ and $\overline{y}$ are positive or negative.

  One solution is to define

  $$A = \{\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y}\}$$

  and then set

  $$[x] \times [y] = [\min A, \max A]$$

- $[x] \div [y]$ For this we can use multiplication after taking the reciprocal of $[y]$.

  Note the special case in which

  $$0 \in [\underline{y}, \overline{y}]$$

  should be treated as divide by zero.

Thus $\frac{1}{[y]} = \begin{cases} \text{undef} & \text{in } 0 \in [\underline{y}, \overline{y}] \\ [\min\{\frac{1}{\underline{y}}, \frac{1}{\overline{y}}\}, \max\{\frac{1}{\underline{y}}, \frac{1}{\overline{y}}\}] & \text{otherwise} \end{cases}$

Then define $[x] \div [y] = [x] \times \frac{1}{[y]}$.

- $\sqrt{[x]}$ when $\underline{x} < 0$ this might be the square root of a negative number, which is imaginary. While it is possible to define rules for the propagation of errors in complex arithmetic, that's outside the scope of the present problem.

Therefore

$$\sqrt{[x]} = \begin{cases} \text{undef} & \text{if } \underline{x} < 0 \\ [\sqrt{\underline{x}}, \sqrt{\overline{x}}] & \text{otherwise} \end{cases}$$

In the case of finite precision arithmetic, it's useful to consider two additional rounding functions.

rdown(x) and rup(x) for $x \in \mathbb{R}$

which respectively round x up or down. Specifically

Let $\mathbb{F} = \{x_* : x_* \text{ is a representable floating point number}\}$

$$\text{rdown}(x) = \max\{x_* \in \mathbb{F} : x_* \leq x\}$$
$$\text{rup}(x) = \min\{x_* \in \mathbb{F} : x_* \geq x\}.$$

After noting that these rounding modes are available on all modern processors, we define

- $[\underline{x}] + [\underline{y}] = [\text{rdown}(\underline{x} + \underline{y}), \text{rup}(\overline{x} + \overline{y})]$
- $[\underline{x}] - [\underline{y}] = [\text{rdown}(\underline{x} - \overline{y}), \text{rup}(\overline{x} - \underline{y})]$
- $[\underline{x}] \times [\underline{y}] = [\text{rdown}(\min A), \text{rup}(\max A)]$

Note that

$$\text{rdown}(\min A) = \min\{\text{rdown}(\underline{x}\,\underline{y}), \text{rdown}(\underline{x}\,\overline{y}), \text{rdown}(\overline{x}\,\underline{y}), \text{rdown}(\overline{x}\,\overline{y})\}$$

so there is no difficulty computing this quantity on the computer

- $[x] \div [y]$ define

$$\frac{1}{[y]} = \begin{cases} \text{undef} & \text{if } 0 \in [\underline{y}, \overline{y}] \\ \left[\min\{\text{rdown}\,\frac{1}{\underline{y}}, \text{rdown}\,\frac{1}{\overline{y}}\}, \max\{\text{rup}\,\frac{1}{\underline{y}}, \text{rup}\,\frac{1}{\overline{y}}\}\right] & \text{otherwise} \end{cases}$$

and then computes $[x] \times \frac{1}{[y]}$ using the previous rule.

- $\sqrt{[x]}$ It may not be the case that round up and round down are available for the square root

In fact its possible the subroutine implementing sqrt(x) does not even follow round to nearest over the entire range of available values for x.

In this case suppose $y*$ represents the approximation computed by the sqrt(x) function and the relative error of this approximation is bounded as

$$\left| \frac{\text{sqrt}(x) - \sqrt{x}}{\sqrt{x}} \right| \leq \varepsilon \quad \text{for all } x \in F.$$

Then $\quad -\varepsilon\sqrt{x} \leq \text{sqrt}(x) - \sqrt{x} \leq \varepsilon\sqrt{x}$

implies $\quad (1-\varepsilon)\sqrt{x} \leq \text{sqrt}(x) \leq (1+\varepsilon)\sqrt{x}$

or that $\frac{\text{sqrt}(x)}{1+\varepsilon} \leq \sqrt{x} \leq \frac{\text{sqrt}(x)}{1-\varepsilon}$

consequently $\text{rdown}\left(\frac{\text{sqrt}(x)}{1+\varepsilon}\right) \leq \sqrt{x} \leq \text{rup}\left(\frac{\text{sqrt}\,x}{1-\varepsilon}\right)$

note that everything here and here can be calculated on the computer

Thus

$$\square[x] = \begin{cases} \text{undef} & \text{if } x < 1 \\ \left[\text{rdown}\left(\frac{\text{sqrt}(x)}{1+\varepsilon}\right), \text{rup}\left(\frac{\text{sqrt}(x)}{1-\varepsilon}\right)\right] & \text{otherwise} \end{cases}$$

Note that other bounds may also be possible for $\square[x]$

3.1 Can *all* matrices $A \in \mathbb{R}^{n \times n}$ be factored $A = LU$? Why or why not?

No, some matrices require row swaps in order to be factored. For example

$$A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

can not be factored into LU because $a_{11} = 0$.

DH 3.3 Factor the following matrix A as a product $A = LU$:

$$\begin{pmatrix} 1 & 2 & 7 \\ 3 & 5 & -1 \\ 6 & 1 & 4 \end{pmatrix}.$$

$$\begin{bmatrix} 1 & 2 & 7 \\ 3 & 5 & -1 \\ 6 & 1 & 4 \end{bmatrix}$$

$r_2 \leftarrow r_2 - 3r_1$

$r_3 \leftarrow r_3 - 6r_1$

$$\begin{bmatrix} 1 & 2 & 7 \\ 0 & -1 & -22 \\ 0 & -11 & -38 \end{bmatrix}$$

$r_3 \leftarrow r_3 - 11r_2$

$$\begin{array}{r} 22 \\ 11 \\ \hline 22 \\ 22 \\ \hline 242 \\ -38 \\ \hline 204 \end{array}$$

$$U = \begin{bmatrix} 1 & 2 & 7 \\ 0 & -1 & -22 \\ 0 & 0 & 204 \end{bmatrix} \qquad L = \begin{bmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ 6 & 11 & 1 \end{bmatrix}$$

```
julia> U=[1 2 7; 0 -1 -22; 0 0 204]
3×3 Matrix{Int64}:
 1   2    7
 0  -1  -22
 0   0  204

julia> L=[1 0 0; 3 1 0; 6 11 1]
3×3 Matrix{Int64}:
 1   0  0
 3   1  0
 6  11  1

julia> L*U
3×3 Matrix{Int64}:
 1  2   7
 3  5  -1
 6  1   4

julia>
```

Checks...

3.11 Show how the LU factorization of $A \in \mathbb{R}^{n \times n}$ can be used to compute the determinant of A.

The cofactor expression for determinant is

$$\det A = \sum_{j=1}^{n} (-1)^{i+j} a_{ij} \det A_{ij}$$

where A_{ij} is the matrix formed by crossing out the i^{th} row and j^{th} column of the matrix A.

Suppose U is an upper triangular matrix. Then

$$U = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ 0 & 0 & \ddots & \vdots \\ 0 & 0 & 0 & u_{nn} \end{bmatrix}$$

In this case $u_{ij} = 0$ for $i > j$.

It follows taking $i = 1$ that

$$\det U = \sum_{j=1}^{n} (-1)^{1+j} u_{1j} \det U_{1j} = u_{11} \det U_{11}$$

Note that U_{11} is

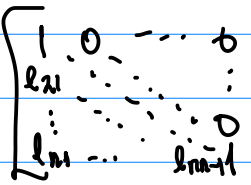
$$U = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ 0 & 0 & \ddots & \vdots \\ 0 & 0 & 0 & u_{nn} \end{bmatrix}$$

→ also an upper triangular matrix...

Therefore $\det U_{11}$ can be computed as $u_{22} \det \begin{bmatrix} u_{33} & \cdots & u_{3n} \\ 0 & \ddots & \vdots \\ 0 & 0 & u_{nn} \end{bmatrix}$

and so on so $\det U = u_{11} u_{22} \cdots u_{nn}$ is the product along the diagonal.

A similar argument implies

$\det h = 1$ where $h =$ 

since there are only 1's on the diagonal.

Thus,

$$\det(A) = \det(LU) = \det(L)\det(U) = u_{11}u_{22}\cdots u_{nn}$$

Therefore to compute the determinant of A, first find the LU factorization and then multiply the diagonal elements of U together to get the answer.

3.12 For numerical stability and generality, we incorporated pivoting into our methods for Gaussian elimination. We can modify our construction of the LU factorization somewhat to incorporate pivoting as well.

(a) Argue that following the steps of Gaussian elimination on a matrix $A \in \mathbb{R}^{n\times n}$ with partial pivoting can be used to write $U = L_{n-1}P_{n-1}\cdots L_2P_2L_1P_1A$, where the P_i's are permutation matrices, the L_i's are lower triangular, and U is upper triangular.

(b) Show that P_i is a permutation matrix that swaps rows i and j for some $j \geq i$. Also, argue that L_i is the product of matrices of the form $I_{n\times n} + c\vec{e}_k\vec{e}_i^{\top}$ where $k > i$.

(c) Suppose $j, k > i$. Show $P_{jk}(I_{n\times n} + c\vec{e}_k\vec{e}_i^{\top}) = (I_{n\times n} + c\vec{e}_j\vec{e}_i^{\top})P_{jk}$, where P_{jk} is a permutation matrix swapping rows j and k.

(d) Combine the previous two parts to show that

$$L_{n-1}P_{n-1}\cdots L_2P_2L_1P_1 = L_{n-1}L'_{n-2}L'_{n-3}\cdots L'_1P_{n-1}\cdots P_2P_1,$$

where $L'_1, \ldots, L'_{n-2}$ are lower triangular.

(e) Conclude that $A = PLU$, where P is a permutation matrix, L is lower triangular, and U is upper triangular.

(f) Extend the method from §3.5.2 for solving $A\vec{x} = \vec{b}$ when we have factored $A = PLU$, without affecting the time complexity compared to factorizations $A = LU$.

(a) Elimination with partial pivoting consists of a sequence of elimination steps of the form $r_i \leftarrow r_i - \alpha r_j$ where $i > j$ followed by row swaps $r_i \leftrightarrow r_j$.

(b) Since $A = IA$, the elimination step $r_i \leftarrow r_i - \alpha r_j$ corresponds to multiplication by a matrix formed by performing the elimination step on the identity matrix. In particular, the matrix for

$$[r_i \leftarrow r_i - \alpha r_j] = I - \alpha e_i e_i^T$$

is lower triangular since $i > j$ and the matrix

$$[r_i \leftrightarrow r_j] = I - e_i e_i^T + e_j e_i^T - e_j e_j^T + e_i e_j^T$$

is a permutation matrix that swaps row i with row j.

The reduction to row echelon form results in the upper triangular matrix U obtained as

$$[r_n \leftarrow r_n - a_{n,n-1} r_{n-1}] \cdots [r_2 \leftrightarrow r_k] [r_n \leftarrow r_n - a_{n1} r_1] \cdots [r_2 \leftarrow r_2 - a_{21} r_1] [r_i \leftrightarrow r_j] A = U$$

Since the inverse of $r_i \leftarrow r_i - \alpha r_j$ is $r_i \leftarrow r_i + \alpha r_j$ and also triangular, each of these matrices can be inverted in order and placed on the other side to obtain

$$A = [r_i \leftrightarrow r_j] [r_2 \leftarrow r_2 + a_{21} r_1] \cdots [r_n \leftarrow r_n + a_{n1} r_1] [r_2 \leftrightarrow r_k] \cdots [r_n \leftarrow r_n + a_{n,n-1} r_{n-1}] U.$$

P_1 — L_1 (elimination steps between row swap) — P_2 — $\cdots$ — L_{n-1}

Now group the elimination steps between each row swap together

Since the product of lower triangular matrices is again lower triangular Then each of the L_j's are lower triangular. We obtain

$$A = P_1 L_1 P_2 \cdots L_{n-1} U$$

(c) If $j, k > i$ then $j \neq i$ and $k \neq i$. Now, since performing a row swap followed by an elimination step is the same as performing the elimination step on the row that was swapped first and then swapping it, then the sequence

$r_k \leftarrow r_k + c r_i$ followed by $r_j \leftrightarrow r_k$

is the same as

$r_j \leftrightarrow r_k$ followed by $r_j \leftarrow r_j + c r_i$

Written in matrix notation, this means

$$P_{jk} (I + c e_k e_i^T) = (I + c e_j e_i^T) P_{jk}$$

Note that the order of the operations represented by the matrix multiplication read from right to left as matrix multiplication really means composition of linear functions.

(e) One can switch the order of the L and P matrices using the result of (c) to group all the L matrices together.

Thus $A = P_1 L_1 P_2 \cdots L_{n-1} U = P_1 P_2 \cdots P_{n-1} L_1' L_2' \cdots L_{n-1}' U$

again lower triangular

(c) Since multiplication of lower triangular matrices are again lower triangular than $L = L_1' L_2' \cdots L_{n-1}'$ is lower triangular.

Since $P = P_1 P_2 \cdots P_{n-1}$ is also a permutation, then

$$A = PLU.$$

(f) To solve $Ax = b$ use the fact that $P^T = P^{-1}$ and simplify $PLUx = b$ to obtain $LUx = P^T b$.

At this point on can solve

$$\begin{cases} Ly = P^T b \\ Ux = y \end{cases}$$

as before using forward and back substitution.