Task 1:

a) Lager en funksjon som returnerer funksjonen: f(x) = -0.69x^2 + 1.3x + 0.42

Bruker np.arange(0, 2.5, 0.1) for å få x-verdier fra 0 til 2.5 med 0.1 mellomrom.

regner ut yverdier med funksjonen. og plotter med px.line(x,y)

b) Fibonacci: begynner fibo med fib=[0,1], n =25. for antall iterasjoner.

lager en for loop for å beregne fibonacci nummerene: fib.append(fib[i-2]+fib[i-2])

Dette tar tallet to tilbake i arrayet/listen og tallet en tilbake i listen og plusser det sammen og legger det i nåværende plass i listen altså i. gjør dette frem til n+2. Dette er da y verdiene også lager jeg x verdiene som bare er 1->25/26.

Task 2:

Skriver opp en matrise på formen under.

Et bilde som inneholder tekst, skjermbilde, Font, design

Automatisk generert beskrivelse

Ganger den inverse matrisen med y verdiene for å finne a vektoren, som da inneholder 2-gradspolynomet som går gjennom alle punktene.

Et bilde som inneholder Font, tekst, klokke, design

Automatisk generert beskrivelse

Tar dette på kalkulatoren of får  
 =

I python blir dette en array med 3 arrays, der arrayene [0],[1] og [2] blir de tre i matrisen.

Også har vi enda en array b som er da blir vektor Y.

I pthon bruker vi funksjonen fram numpy linalg.solve  
np.linalg.solve(a,b) og får svarene i ett array med 3 index, de array[0] = osv…

Task 3

Oppgaven er å finne a,b og c så de oppfyller oppgavens kriterier som er :  
1. hvis b er partall: a= c+11 og oddetall: a = 2c-129

2. b = (a\*b) modulo 2377

3. c = summen av (b-7\*k) når k øker med en for hver iterasjon, opptil a-1 ganger. +142

4. alle må være under 450 i verdi og over 0 i verdi.

Løsningen krever at vi gjør det i riktig rekkefølge for å finne en løsning.

1. må generere en random C først, for å kunne kalkulere en A
2. Kalkuler A uten å vite B. Alltid sjekk om A er mellom 450 og 0
3. Kalkuler B med A og C, deretter sjekk om B blir oddetall eller partall.
4. Om B er partall, bruk A\_even, ellers bruk A\_odd.
5. Så kalkulerer du C sin condition.
6. Tilslutt sjekker du om C sin condition er lik Den genererte C-en. Hvis det, så hjar vi funnet en løsning.

Task4

**Min forklaring:**

* **De tre hovedfunksjonene:**
  + **Sample\_theta(size) = curr\_theta**
    - Denne funksjonen henter bare 9 random verdier(parametere) mellom -4 og 4.
  + **Predict (xs, curr\_theta) = y\_hat**
    - Denne funksjonen tar inn de gitte xs verdiene, og kalkulerer y\_hat med de curr\_theta (de 9 randomiserte verdiene) som er y verdiene til de gitte xs verdiene. Punktene vil bli (xs [0], y\_hat [0])
  + **Get\_loss (y\_hat, ys) = curr\_loss**
    - Denne funksjonen kalkulerer mse loss, ut ifra de y verdiene du har fått ved hjelp av de 9 random verdiene og de gitte ys verdiene, som er «løsningen».
    - Losset sier noe om hvor god tilnærming vår predikerte modell til den faktiske modellen.
* **De tre for-løkkene:**
* **Alt som er grønt er hio**
* **Alt som er i blått er egt ikke med i fortuna algoritmen.**
* **Loop [100]:** Kjører hele programmet 100 ganger, for å få en average for å se hvor bra programmet fungerer
  + **Best\_Loss\_100Run:** Lagrer laveste loss fra 100 runs
  + **Best\_Theta\_100Run:** Lagrer de beste randomiserte parameterne fra 100 runs
  + **Loop [10]**
  + **Total\_loss1 += Best\_Loss\_100Run:** Denne kjøres etter Loop [10] og Loop [10000].Den bareplusser på det laveste losset fra forrige run.
    - **Loop [10]:** Antall resets \*
      * **best\_restart\_theta = sample\_theta (9)**
        + Resetter og/eller Lager 9 random parametere
      * **y\_hat = predict (xs, best\_restart\_theta)**
        + Predikerer y-verdier basert på de 9 random parametere
      * **best\_restart\_loss = get\_loss (y\_hat, ys)**
        + Berenger loss fra de predikerte y-verdiene og løsningen
      * **best\_theta = None**
        + Definerer best\_theta
      * **best\_loss = float('inf')**
        + Definerer best\_loss og setter som høyest mulig.
      * **Loop [10 000]**
        + **delta\_theta = sample\_theta(n\_params) \* 0.01**

lagrer 10% av en ny random theta.

* + - * + **temp\_theta = best\_restart\_theta + delta\_theta**

tar current theta som blir oppdatert til å være den beste for hver restart og plusser på delta\_theta, som er steglengden.

* + - * + **temp\_theta = np. clip (temp\_theta, -4, 4)**

Denne linjen er bare for å sjekke at ingen av parameterne i theta kommer over 4 eller under -4, hvis de gjør det blir de satt til enten 4 eller -4.

* + - * + **temp\_y\_hat = predict (xs, temp\_theta)**

Predikerer y-verdier basert på vår nye midlertidige theta.

* + - * + **temp\_loss = get\_loss (temp\_y\_hat, ys)**

Berenger midlertidig loss ut ifra de midlertidige predikerte y-verdiene

* + - * + **if temp\_loss < best\_restart\_loss:**

Sjekker om den midlertidige lossen er mindre enn current loss, som skal være den beste loss i denne restarten.

**best\_restart\_theta = temp\_theta**

Setter best\_restart\_theta til temp\_theta, for å beholde de beste parametere for denne restarten.

**best\_restart\_loss = temp\_loss**

Setter til ny laveste loss. Som tilhører ny beste theta.

**if best\_restart\_loss < best\_loss:**

Hvis den nye beste lossen er bedre enn vår nåværende all time beste loss så …

**best\_loss = best\_restart\_loss**

Setter best loss til ny best loss.

**best\_theta = best\_restart\_theta. copy ()**

Setter best theta til ny best theta.

* + - * **Loop [10 000]**
      * **if best\_loss < best\_loss\_100run:**

Sjekker om best loss funnet etter hver restart/loop[10] iterasjon er bedre enn best loss fra de andre runsene.

* + - * + **best\_loss\_100run = best\_loss**

**Oppdaterer best loss i Loop [100].**

* + - * + **best\_theta\_100run = best\_theta. copy ()**

**Oppdaterer best theta i Loop [100].**

**De 3 For-løkkene**

1. **Ytterste løkke - Kjøringer (for run in range(num\_runs)):**
   * Denne løkken kjører optimaliseringsprosessen num\_runs ganger (20 ganger i dette tilfellet).
   * Målet er å evaluere ulike startpunkter og akkumulere den beste gjennomsnittlige ytelsen.
   * Etter hver kjøring beregner den det beste tapet funnet under den kjøringen (best\_loss\_run).
2. **Mellomste løkke - Restarter (for restart in range(num\_restarts)):**
   * For hver kjøring utføres num\_restarts restarter (10 restarter her).
   * Hver restart starter med en ny tilfeldig initialisert theta for å utforske forskjellige områder i parameterrommet.
   * I hver restart blir de nåværende parameterne (best\_restart\_theta) justert for å finne et forbedret tap.
3. **Innerste løkke - Iterasjoner (for \_ in range(iterations\_per\_restart)):**
   * Denne løkken justerer gjeldende parametere (best\_restart\_theta) iterativt for å forbedre tapet, og kjøres iterations\_per\_restart ganger.
   * I hver iterasjon:
     + En liten tilfeldig forstyrrelse (delta\_theta) genereres.
     + Denne forstyrrelsen legges til best\_restart\_theta for å produsere et kandidatsett av parametere (candidate\_theta).
     + candidate\_theta blir deretter klippet slik at hver parameter holder seg innenfor området [-4, 4].
     + Tapet beregnes for candidate\_theta og sammenlignes med det nåværende tapet (best\_restart\_loss).
     + Hvis candidate\_loss er bedre (lavere), oppdateres best\_restart\_theta til candidate\_theta, og også det nåværende tapet oppdateres.

**Hvordan theta oppdateres**

* **Steglengdejustering:** I hver iterasjon genereres en liten justering (delta\_theta) ved hjelp av sample\_theta(n\_params) multiplisert med step\_size (0.01), som gir en liten endring i hver parameter.
* **Beregning av ny theta:**
  + candidate\_theta = best\_restart\_theta + delta\_theta lager et nytt sett med parametere.
  + Disse parameterne blir deretter klippet for å holde alle verdier innenfor området [-4, 4] for å sikre at de forblir gyldige.
* **Akseptkontroll:**
  + Hvis de nye parameterne (candidate\_theta) resulterer i et lavere tap (candidate\_loss), aksepteres de, og best\_restart\_theta og best\_restart\_loss oppdateres tilsvarende.
  + Dette er kjernen i **hill-klatring**-prosessen, der kun endringer som forbedrer tapet blir akseptert.

Denne iterative prosessen hjelper med å finne det beste settet med parametere (theta) for å tilpasse de gitte datapunktene (xs og ys).