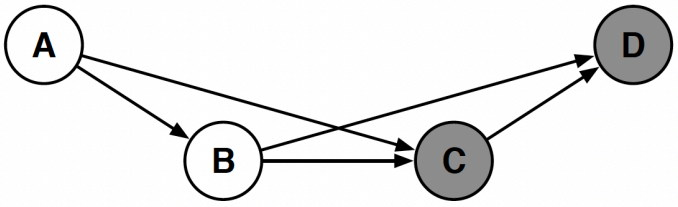
6.1 Survey

Filled out. A lot of great movies there.

6.2 EM Algorithm



a) Posterior probability

We consider the BN shown above, with observed nodes C and D and hidden nodes A and B.

Compute $P(a,b|c,d)$ in terms of $P(a)$, $P(b|a)$, $P(c|a,b)$ and $P(d|b,c)$.

$$P(a,b|c,d) \overset{\text{P.R.}}{=} \frac{P(a,b,c,d)}{P(c,d)}$$

$$\overset{\text{marg.}}{=} \frac{P(a,b,c,d)}{\sum_a \sum_b P(a=a,b=b,c,d)}$$

$$\overset{\text{P.R.}}{=} \frac{P(a)P(b|a)P(c|a,b)P(d|a,b,c)}{\sum_a \sum_b P(a=a)P(b=b|a=a)P(c|a=a,b=b)P(d|a=a,b=b,c)}$$

$$\overset{\text{d-sep}}{=} \frac{P(a)P(b|a)P(c|a,b)P(d|b,c)}{\sum_a \sum_b P(a=a)P(b=b|a=a)P(c|a=a,b=b)P(d|b=b,c)}$$

b) Posterior probability

Compute $P(a|c,d)$ and $P(b|c,d)$ in terms of the answer from a). That is, assume $P(a,b|c,d)$ is given.

$$P(a|c,d) \overset{\text{marg}}{=} \sum_b P(a,b=b|c,d) \quad \text{which is given.}$$

$$P(b|c,d) \overset{\text{marg}}{=} \sum_a P(a=a,b|c,d) \quad \text{which is given.}$$

c) Log-likelihood

We consider a partially complete data set of i.i.d. examples $\{c_t, d_t\}_{t=1}^{T}$ drawn from the joint distribution of the BN.

The log-likelihood of the data set is given by: $\mathcal{L} = \sum_t \log P(C=c_t, D=d_t)$.

Compute this log-likelihood in terms of the CPTs of the BN. We may re-use work from earlier parts of the problem.

$$\mathcal{L} = \sum_t \log P(C=c_t, D=d_t)$$

$$\overset{\text{marg}}{=} \sum_t \log \sum_a \sum_b P(A=a, B=b, C=c_t, D=d_t)$$

$$\overset{\text{P.R.}}{=} \sum_t \log \sum_a \sum_b P(A=a)P(B=b|A=a)P(C=c_t|A=a,B=b)P(D=d_t|A=a,B=b,C=c_t)$$

$$\overset{\text{d-sep}}{=} \sum_t \log \sum_a \sum_b P(A=a)P(B=b|A=a)P(C=c_t|A=a,B=b)P(D=d_t|B=b,C=c_t)$$

d) EM algorithm

Give the EM updates to estimate CPTs that maximize the log-likelihood in c). Express in terms of $P(a,b|c_t,d_t)$, $P(a|c_t,d_t)$, $P(a|c_t,d_t)$, $I(c,c_t)$, $I(d,d_t)$

$$P(A=a) \leftarrow \frac{\sum_t P(a \mid c_t, d_t)}{T}$$

$$P(B=b|A=a) \leftarrow \frac{\sum_t P(a,b \mid c_t, d_t)}{\sum_t P(a \mid c_t, d_t)}$$

$$P(C=c|A=a,B=b) \leftarrow \frac{\sum_t I(c,c_t)\, P(a,b \mid c_t, d_t)}{\sum_t P(a,b \mid c_t, d_t)}$$

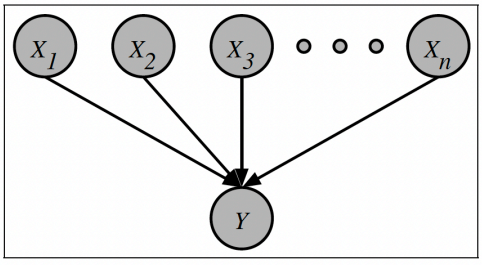
$$P(D=d|B=b,C=c) \leftarrow \frac{\sum_t I(d,d_t)\, I(c,c_t)\, P(b \mid c_t, d_t)}{\sum_t I(c,c_t)\, P(b \mid c_t, d_t)}$$

6.3 EM algorithm for noisy-OR

Consider the belief network on the right, with binary random variables $X \in \{0,1\}^n$ and $Y \in \{0,1\}$ and a noisy-OR conditional probability table (CPT). The noisy-OR CPT is given by:

$$P(Y=1|X) \;=\; 1 - \prod_{i=1}^{n} (1-p_i)^{X_i},$$

which is expressed in terms of the noisy-OR parameters $p_i \in [0,1]$.



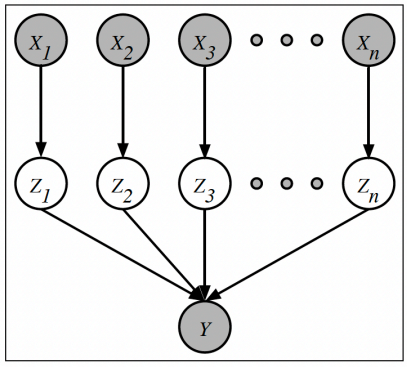
In this problem, you will derive and implement an EM algorithm for estimating the noisy-OR parameters p_i. It may seem that the EM algorithm is not suited to this problem, in which all the nodes are observed, and the CPT has a parameterized form. In fact, the EM algorithm can be applied, but first we must express the model in a different but equivalent form.

Consider the belief network shown to the right. In this network, a binary random variable $Z_i \in \{0,1\}$ intercedes between each pair of nodes X_i and Y. Suppose that:

$$\begin{aligned} P(Z_i=1|X_i=0) &= 0, \\ P(Z_i=1|X_i=1) &= p_i. \end{aligned}$$

Also, let the node Y be *determined* by the logical-OR of Z_i. In other words:

$$P(Y=1|Z) \;=\; \begin{cases} 1 \text{ if } Z_i=1 \text{ for any } i, \\ 0 \text{ if } Z_i=0 \text{ for all } i. \end{cases}$$



a) Show that this "extended" BN defines the same conditional distribution $P(Y|X)$ as the original one. In particular, starting from $P(Y=1|X) = \sum_{z \in \{0,1\}^n} P(Y=1, z|x)$, show that the RHS reduces to the noisy-OR CPT w/ parameters p_i.

$$\sum_{z\in\{0,1\}^n} P(Y=1,z|x) \overset{\text{P.R.}}{=} \sum_{z\in\{0,1\}^n} P(Y=1|z,x)P(z|x)$$

$$\overset{\text{d-sep}}{=} \sum_{z\in\{0,1\}^n} P(Y=1|z)P(z|x)$$

$$= \sum_{z\in\{0,1\}^n} \prod_{i=1}^{n} P(z_i|x_i) \qquad P(Y=1|z)=1 \text{ if } z_i=1,\ 0 \text{ else}$$

$$= 1 - \prod_{i=1}^{n} P(z_i=0|x)$$

$$= 1 - \prod_{i=1}^{n} (1-p_i)^{x_i}$$

QED

Consider estimating the noisy-OR parameters p_i to maximize the (conditional) likelihood of the observed data. The (normalized) log-likelihood in this case is given by:

$$\mathcal{L} = \frac{1}{T}\sum_{t=1}^{T} \log P(Y = y^{(t)} | X = \vec{x}^{(t)}),$$

where $(\vec{x}^{(t)}, y^{(t)})$ is the tth joint observation of X and Y, and where for convenience we have divided the overall log-likelihood by the number of examples T. From your result in part (a), it follows that we can estimate the parameters p_i in either the original network or the extended one (since in both networks they would be maximizing the same equation for the log-likelihood).

Notice that in the extended network, we can view X and Y as observed nodes and Z as hidden nodes. Thus in this network, we can use the EM algorithm to estimate each parameter p_i, which simply defines one row of the "look-up" CPT for the node Z_i.

b) For joint observations $x \in \{0,1\}^n$ and $y \in \{0,1\}$, use Bayes to show that $P(Z_i=1, X_i=1 | X=x, Y=y) = \dfrac{y x_i p_i}{1 - \prod_j (1-p_j)^{x_j}}$.

We can take $\mathbb{I}(x_i=1, x)$ out x_i

$$P(Z_i=1, X_i=1 | X=x, Y=y) = x_i P(Z_i=1 | X=x, Y=y)$$

Bayes
$$= x_i \frac{P(Y=y | Z_i=1, X=x) P(Z_i=1 | X=x)}{P(Y=y | X=x)}$$

d-sep 1
$$= x_i \frac{P(Y=y | Z_i=1) P(Z_i=1 | X=x)}{P(Y=y | X=x)}$$

When $y=0 \Rightarrow P=0$
$y=1 \Rightarrow P(Y=1|Z_i=1)=1$
$$= \frac{x_i y P(Z_i=1 | X=x)}{P(Y=1 | X=x)}$$

if $x=0$
$P(Z_i=1|X=0)=0$
$\Rightarrow P=0$.
else $P(Z_i=1|X=1)=p_i$
$$= \frac{x_i y p_i}{P(Y=1 | X=x)}$$

$$= \frac{y x_i p_i}{1 - \prod_j (1-p_j)^{x_j}}$$

because $P(Y=1 | X=x) = 1 - \prod_{j=1}^{n} (1-p_j)^{x_j}$ as found in a). QED.

c) For the data set $\{\vec{x}^{(t)}, y^{(t)}\}_{t=1}^{T}$, show that the EM update for p_i is given by $p_i \leftarrow \frac{1}{T_i}\sum_t P(Z_i=1, X_i=1 | X=x^{(t)}, Y=y^{(t)})$, where T_i is num of examples where $X_i=1$.

$$p_i = P(Z_i=1 | X_i=1) \longleftarrow \frac{\sum_t P(Z_i=1, X_i=1 | X=x^{(t)}, Y=y^{(t)})}{\sum_t P(X_i=1 | X=x^{(t)}, Y=y^{(t)})} = \frac{\sum_t P(Z_i=1, X_i=1 | X=x^{(t)}, Y=y^{(t)})}{T_i}$$

QED

Download the data files on the course web site, and use the EM algorithm to estimate the parameters p_i. The data set[1] has $T = 267$ examples over $n = 23$ inputs. To check your solution, initialize all $p_i = 0.05$ and perform 256 iterations of the EM algorithm. At each iteration, compute the log-likelihood shown in part (b). (If you have implemented the EM algorithm correctly, this log-likelihood will always increase from one iteration to the next.) Also compute the number of mistakes $M \leq T$ made by the model at each iteration; a mistake occurs either when $y_t = 0$ and $P(y_t = 1|\vec{x}_t) \geq 0.5$ (indicating a false positive) or when $y_t = 1$ and $P(y_t = 1|\vec{x}_t) \leq 0.5$ (indicating a false negative). The number of mistakes should generally decrease as the model is trained, though it is not guaranteed to do so at each iteration. Complete the following table:

d) Complete the table.

iteration	number of mistakes M	log-likelihood $\mathcal{L}$
0	175	-0.95809
1	56	-0.49592
2	43	-0.40822
4	42	-0.36461
8	44	-0.34750
16	40	-0.33462
32	37	-0.32258
64	37	-0.31483
128	36	-0.31116
256	36	-0.31016

e) Source code

```
f = open("noisyOrX.txt", "r")
X = np.array([line.rstrip(' \n').split(' ') for line in f]).astype(int)
f.close()

f = open("noisyOrY.txt", "r")
Y = np.array([line.rstrip("\n") for line in f]).astype(int)
f.close()
```

✓ 0.2s

```
T = len(X)
n_iter = 256
n = 23
p_i = 0.05*np.ones(n)

# Find T_i (number of times x=1)
T_i = np.zeros(n)
for i in range(len(X)):
    for j in range(n):
        if X[i][j] == 1:
            T_i[j] += 1
```

✓ 0.2s

```
L = np.zeros(n_iter+1)
M = np.zeros(n_iter+1)
```

✓ 0.3s

```
def EM_algorithm(p_i, T, n, T_i, n_iter, log, M, X, Y):
    for k in range(n_iter+1):

        # Log likelihood
        for t in range(T):
            term1 = 0
            term2 = 1
            for i in range(n):
                term1 += X[t][i]*np.log(1-p_i[i])
                term2 *= np.power((1-p_i[i]),X[t][i])
            L[k] += ((1-Y[t])*term1 + Y[t]*np.log(1-term2))
        L[k] = L[k]/T

        # Mistakes
        prob = np.zeros(T)
        for t in range(T):
            tp = 1
            for i in range(n):
                tp *= np.power((1-p_i[i]),X[t][i])
            prob[t] = 1-tp
            if ((Y[t]==1 and prob[t]<=0.5) or (Y[t]==0 and prob[t]>=0.5)):
                M[k] += 1

        # E step
        posterior = np.zeros((T,n))
        for t in range(T):
            denom_prod = 1
            for i in range(n):
                denom_prod *= np.power((1-p_i[i]),X[t][i])
            for j in range(n):
                posterior[t][j] = Y[t]*X[t][j]*p_i[j]/(1-denom_prod)

        # M step
        for i in range(n):
            x = 0
            for t in range(T):
                x += posterior[t][i]
            p_i[i] = (1/T_i[i])*x
    return M, L
```

✓ 0.4s

```
M, L = EM_algorithm(p_i, T, n, T_i, n_iter, log, M, X, Y)
```

✓ 30.4s

```
print(f"Iteration: 0 \t Mistakes: {M[0]} \t Log: {round(L[0],5)}")
print(f"Iteration: 1 \t Mistakes: {M[1]} \t Log: {round(L[1],5)}")
print(f"Iteration: 2 \t Mistakes: {M[2]} \t Log: {round(L[2],5)}")
print(f"Iteration: 4 \t Mistakes: {M[4]} \t Log: {round(L[4],5)}")
print(f"Iteration: 8 \t Mistakes: {M[8]} \t Log: {round(L[8],5)}")
print(f"Iteration: 16 \t Mistakes: {M[16]} \t Log: {round(L[16],5)}")
print(f"Iteration: 32 \t Mistakes: {M[32]} \t Log: {round(L[32],5)}")
print(f"Iteration: 64 \t Mistakes: {M[64]} \t Log: {round(L[64],5)}")
print(f"Iteration: 128 \t Mistakes: {M[128]} \t Log: {round(L[128],5)}")
print(f"Iteration: 256 \t Mistakes: {M[256]} \t Log: {round(L[256],5)}")
```

✓ 0.2s

```
Iteration: 0    Mistakes: 175.0         Log: -0.95809
Iteration: 1    Mistakes: 56.0          Log: -0.49592
Iteration: 2    Mistakes: 43.0          Log: -0.40822
Iteration: 4    Mistakes: 42.0          Log: -0.36461
Iteration: 8    Mistakes: 44.0          Log: -0.3475
Iteration: 16   Mistakes: 40.0          Log: -0.33462
Iteration: 32   Mistakes: 37.0          Log: -0.32258
Iteration: 64   Mistakes: 37.0          Log: -0.31483
Iteration: 128  Mistakes: 36.0          Log: -0.31116
Iteration: 256  Mistakes: 36.0          Log: -0.31016
```

6.4 Auxiliary function

In class we derived an auxiliary function for maximizing the log-likelihood in BNs with hidden variables. In this problem we will derive an auxiliary function for minimizing a simpler function that is nearly quadratic near its minimum, but nearly linear far away from its minimum.

a) Consider the function $f(x) = \log\cosh(x)$. Show that the minimum occurs at $x = 0$.

We find its derivative and set it to zero to find its minimum.

$$f'(x) = \frac{1}{\cosh(x)} \cdot \sinh(x)$$

$$= \frac{\frac{e^x - e^{-x}}{2}}{\frac{e^x + e^{-x}}{2}} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = 0$$

$$\Rightarrow e^x - e^{-x} = 0 \iff e^x = e^{-x}$$

$$\Rightarrow \underline{\underline{x = 0}}$$ ■

b) Show that $f''(x) \leq 1 \quad \forall x$.

We know that $f'(x) = \frac{\sinh(x)}{\cosh(x)} = \tanh(x)$

$$\Rightarrow f''(x) = \operatorname{sech}^2(x) = \left(\frac{2}{e^x + e^{-x}}\right)^2$$

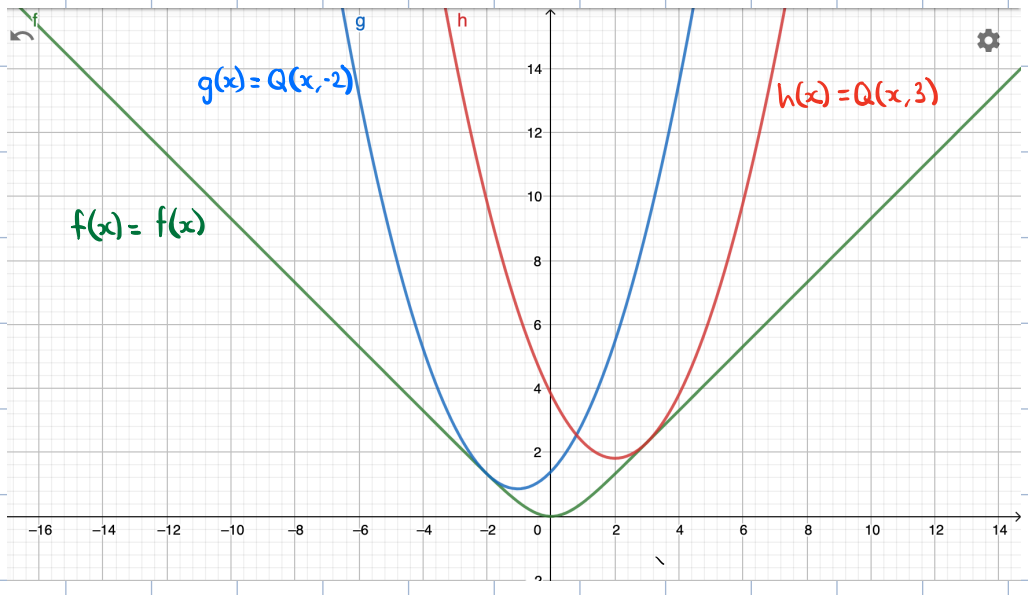
We look at the term inside the parenthesis.

When $x \to 0$, we get $\frac{2}{e^0 + e^{-0}} = \frac{2}{2} = 1$, ← This is max of function. while if we let $x \to \infty$, we would get $\frac{2}{e^\infty + \frac{1}{e^\infty}} \to 0$.

Thus, our extremes are 0 and 1, and if we square it, we will still be in the region $(0, 1]$.

Hence, $f''(x) \leq 1 \quad \forall x$ QED.

c) Consider the function $Q(x,y) = f(y) + f'(y)(x-y) + \frac{1}{2}(x-y)^2$. Plot $f(x)$, $Q(x,-2)$ and $Q(x,3)$ as a function of x.



d) Prove that $Q(x,y)$ is an auxiliary function for $f(x)$. Show that it satisfies (i) $Q(x,x) = f(x)$ and (ii) $Q(x,y) \geq f(x)$.

(i) $Q(x,x) = f(x) + f'(x)(x-x) + \frac{1}{2}(x-x)^2 = f(x)$. OK.

(ii) $f(x) = f(y) + \int_y^x du \left[f'(y) + \int_y^u dv\, f''(v) \right]$

$f''(v) \leq 1$

$\leq f(y) + \int_y^x du \left[f'(y) + \int_y^u dv \right]$

$= f(y) + \int_y^x \left[f'(y) + u - y \right] du$

$= f(y) + \left[f'(y)u + \frac{u^2}{2} - yu \right]_y^x$

$= f(y) + \left(xf'(y) + \frac{x^2}{2} - xy \right) - \left(yf'(y) + \frac{y^2}{2} - y^2 \right)$

$= f(y) + f'(y)(x-y) + \frac{1}{2}(x^2 - 2xy + y^2)$

$= f(y) + f'(y)(x-y) + \frac{1}{2}(x-y)^2 = Q(x,y)$

$\Rightarrow \quad Q(x,y) \geq f(x)$ OK.

$\Rightarrow$ $Q(x,y)$ is an auxiliary function for $f(x)$. QED

e) Derive the form of the update rule $x_{n+1} = \underset{x}{\operatorname{argmin}}\, Q(x, x_n)$.

$\frac{\partial Q(x,x_n)}{\partial x} = f'(x_n) + x - x_n = 0$

$\Rightarrow x_{n+1} = x_n - f'(x_n) = x_n - \tanh(x_n)$

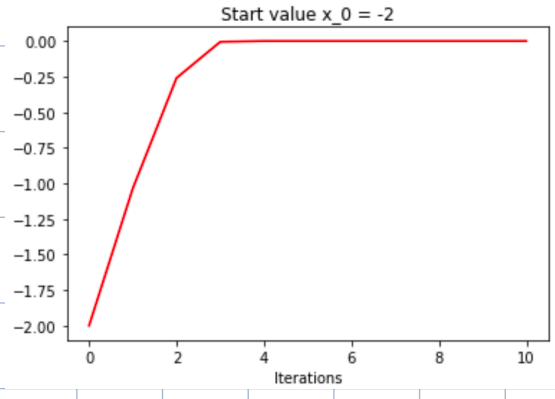
f) Write a simple program to show that x_{n+1} converges numerically for $x_0 = -2$ and $x_0 = 3$. Turn in code and plots.

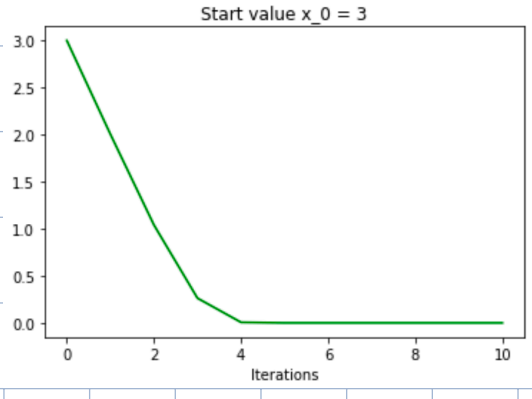
```python
import matplotlib.pyplot as plt
import math as m

def update(x_0,n_iter):
    updates = []
    updates.append(x_0)
    x = x_0
    for i in range(n_iter):
        x = x - m.tanh(x)
        updates.append(x)
    return updates
```

```python
# Start value x_0 = -2
updates = update(x_0=-2,n_iter=10)
plt.plot(updates,'-r')
plt.title("Start value x_0 = -2")
plt.xlabel("Iterations")
plt.show()

# Start value x_0 = 3
updates = update(x_0=3,n_iter=10)
plt.plot(updates, '-g')
plt.title("Start value x_0 = 3")
plt.xlabel("Iterations")
plt.show()
```



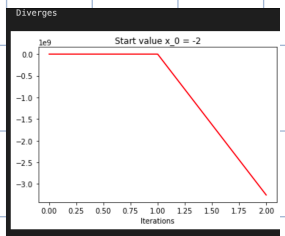


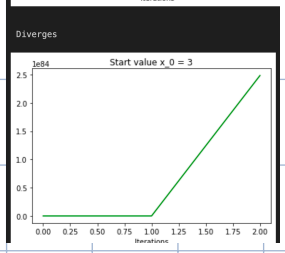
g) Repeat e) and f) using $x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}$. What happens and why? Determine upper bound on $|x_0|$ so that Newton's method converges.

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)} = x_n - \frac{\tanh(x_n)}{\operatorname{sech}^2(x_n)}$$

$$= x_n - \sinh(x_n)\cosh(x_n)$$

$$= x_n - \frac{1}{2}\sinh(2x_n)$$





We observe that the updates diverge rapidly. Sech(x) goes to zero quickly so we will $\to \infty$ fast.

```python
def update_g(x_0,n_iter):
    updates = []
    updates.append(x_0)
    x = x_0
    for i in range(n_iter):
        try:
            x = x - (1/2)*m.sinh(2*x)
            updates.append(x)
        except OverflowError:
            print("Diverges")
            break
    return updates
```

```python
# Start value x_0 = -2
updates = update_g(x_0=-2,n_iter=10)
plt.plot(updates,'-r')
plt.title("Start value x_0 = -2")
plt.xlabel("Iterations")
plt.show()

# Start value x_0 = 3
updates = update_g(x_0=3,n_iter=10)
plt.plot(updates, '-g')
plt.title("Start value x_0 = 3")
plt.xlabel("Iterations")
plt.show()
```

To solve this issue, we use the constraint $|x_1| < |x_0|$

$|x_0 - \frac{1}{2}\sinh(2x_0)| < |x_0|$

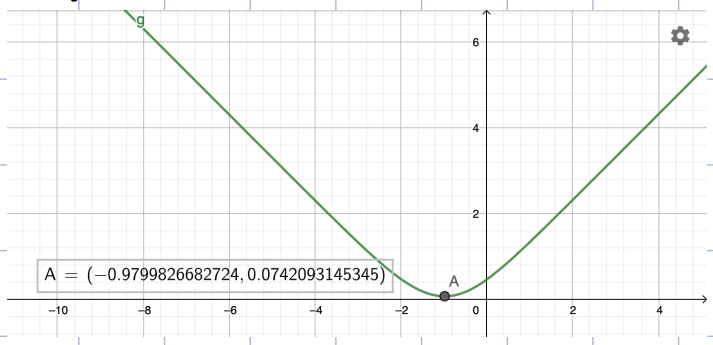
If $x_0 > \frac{1}{2}\sinh(2x_0) > 0$: $x_0 - \frac{1}{2}\sinh(2x_0) < x_0 \Rightarrow \frac{1}{2}\sinh(2x_0) > 0 \Rightarrow x_0 > 0$

If $\frac{1}{2}\sinh(2x_0) > x_0 > 0$: $\frac{1}{2}\sinh(2x_0) - x_0 < x_0 \Rightarrow 0 < x_0 < 1.0866$

We have similar results for $x<0$, so we can yield

$$\underline{|x_0| < 1.0866}$$

h) Plot $g(x) = \frac{1}{10}\sum_{k=1}^{10} \log\cosh\left(x + \frac{2}{\sqrt{k}}\right)$. Is it simple to find the exact minimum?



After the 7th decimal.

No, it is hard to find the exact minimum. Just by moving the plot above, the minimum changes a tiny bit. (Irrational numbers)

i) Consider the function $R(x,y) = g(y) + g'(y)(x-y) + \frac{1}{2}(x-y)^2$. Prove that $R(x,y)$ is an auxiliary function for $g(x)$.

$R(x,x) = g(x) + g'(x)(x-x) + \frac{1}{2}(x-x)^2 = g(x)$ OK

$$g(x) = g(y) + \int_y^x du \left[g'(y) + \int_y^u dv\, g''(v) \right]$$

$g''(x) \leq 1$

$$\leq g(y) + \int_y^x du \left[g'(y) + \int_y^u dv \right]$$

$$= g(y) + \int_y^x [g'(y) + u - y]\, du$$

$$= g(y) + \left[g'(y)u + \frac{u^2}{2} - yu \right]_y^x$$

$$= g(y) + \left(xg'(y) + \frac{x^2}{2} - xy \right) - \left(yg'(y) + \frac{y^2}{2} - y^2 \right)$$

$$= g(y) + g'(y)(x-y) + \frac{1}{2}(x^2 - 2xy + y^2)$$

$$= g(y) + g'(y)(x-y) + \frac{1}{2}(x-y)^2 = R(x,y)$$

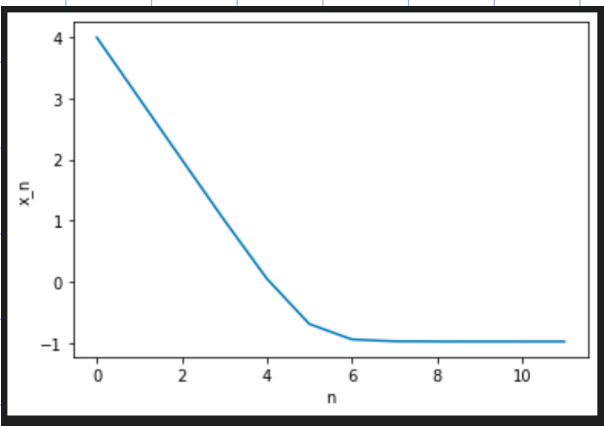
$\Rightarrow R(x,y) \geq g(x) \Rightarrow R(x,y)$ is an auxiliary function for $g(x)$. QED

j) Derive the form of the update rule $x_{n+1} = \operatorname{argmin}_x R(x, x_n)$.

$$\frac{\partial R(x,x_n)}{\partial x} = g'(x_n) + x - x_n = 0$$

$$\Rightarrow x_{n+1} = x_n - g'(x_n) = x_n - \frac{1}{10} \sum_{k=1}^{10} \tanh\left(x_n + \frac{2}{\sqrt{k}}\right)$$

k) Use the update rule to locate min of $g(x)$ to 4 significant digits. Turn in source code as well as plot of x_n vs. n.




```
Minimum value of g(x) is 0.07421 for x = -0.9799819756927353.
```

```python
import numpy as np

def update_k(x_0):
    updates = []
    updates.append(x_0)
    x = x_0
    diff = 1
    while abs(diff)>0:
        x_prev = round(x,5)
        add = 0
        for k in range(1,11):
            add += m.tanh(x+(2/np.sqrt(k)))
        x = x - (1/10)*add
        updates.append(x)
        x_now = round(x,5)
        diff = x_now-x_prev
    return updates

def g(x):
    add = 0
    for k in range(1,11):
        add += np.log(m.cosh(x+(2/np.sqrt(k))))
    return (1/10)*add
```

✓ 0.2s

```python
updates = update_k(4)
minx = updates[-1]
plt.plot(updates)
plt.ylabel("x_n")
plt.xlabel("n")
plt.show()
min_g = g(minx)
print(f"Minimum value of g(x) is {round(min_g,5)} for x = {minx}.")
```