Aritmética finita y aproximaciones

Computación 2021

FaMAF

7 abr. 2021

Contenidos

Códigos

Códigos de la clase en:

https://github.com/mlares/compuprof2021

Aproximaciones

Evaluar un polinomio

Supongamos que tenemos un polinomio

$$P(x) = 2x^4 + 3x^3 - 3x^2 + 5x - 1$$

y lo queremos evaluar en un valor de x.

¿Cuál es la mejor forma de hacerlo?

Aquí "mejor" quiere decir más eficiente, usando la menor cantidad de operaciones posibles.

¿Cuántas operaciones lleva esta ecuación?

$$P(x) = 2 \cdot x \cdot x \cdot x \cdot x \; + \; 3 \cdot x \cdot x \cdot x \; - \; 3 \cdot x \cdot x \; + \; 5 \cdot x \; - \; 1$$

Método de Horner

Teorema

Sea un polinomio de grado n dado por

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$$

y sean $b_n = a_n$, *y* $b_k = a_k + b_{k+1} x_0$ *para* $k = 0, 1, \ldots, n-1$

Entonces, $P(x_0) = b_0$ *y además* $P(x) = (x - x_0)Q(x) + b_0$
con

$$Q(x) = b_n x^{n-1} + b_{n-1} x^{n-2} + \ldots + b_2 x + b_1$$

Otra ventaja es que se puede calcular la derivada del polinomio fácilmente:

$$P'(x) = Q(x) + (x - x_0 Q'(x)), \qquad P'(x_0) = Q(x_0)$$

Polinomios en la PC

Evaluar polinomios es una tarea muy importante porque permiten evaluar otras funciones por medio de aproximaciones. Si bien la suma es una de las operaciones más simples en aritmética, en una computadora es bastante difícil hacerlo BIEN.

En particular, los polinomios de Taylor se usan mucho en análisis numérico.

Repaso: series de Taylor

Teorema

Sea una función $f \in C^n[a,b]$, con derivada $f^{(n+1)}$ finita en el intervalo $[a,b]$, y sea $x_0 \in [a,b]$.

Entonces, para cada $x \in [a,b]$ existe un valor $\xi(x)$ entre x_0 y x tal que

$$f(x) = P_n(x) + R_n(x)$$

donde

$$P_n(x) = \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k$$

y

$$R_n(x) = \frac{f^{(n+1)}(\xi(x))}{(n+1)!}(x - x_0)^{n+1}$$

Polinomios de Taylor

El polinomio $P_n(x)$ es el polinomio de Taylor, y su límite para $n \to \infty$ es la serie de Taylor. El término $R_n(x)$ es el error de truncamiento.

Tenemos que pensar ahora en dos problemas:

- **El error que surge de utilizar un número finito de términos.**
- **El error que surge de representar los números con una aritmética finita.**

▷ *e.g.: Computadora vs. matemática*

▷ e.g.: Casos en los que la igualdad analítica no se mantiene en una computadora:

- $(\sqrt{3})^2 = 3$
- $0.3 - 0.1 = 0.2$
- $1 + 2^{-53} > 1$
- $1 + 2^{-53} + 2^{-53} = 2^{-53} + 2^{-53} + 1$

¿Qué tipo de error estamos cometiendo? ¿Truncamiento? ¿Redondeo?

El error en las restas de números similares

Para tener problemas al hacer operaciones de sumas y restas no hace falta trabajar con muchos términos, puede pasar con pocos términos cuando hacemos restas de números muy parecidos.
En general hay que evitar restar cantidades que den un valor próximo a cero.

▷ e.g.: Sabemos que podemos calcular las raíces del polinomio $ax^2 + bx + c$ mediante las fórmulas

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}, \quad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}.$$

Notar que si b^2 tiene una magnitud mucho mayor a la de $4ac$ entonces tendremos mucho error al calcular x_1 si b es positivo (x_2 si b es negativo).

```
a=1; b=-62.1; c=1
x1=(-b+m.sqrt(b**2-4*a*c))/(2*a)
x2=(-b-m.sqrt(b**2-4*a*c))/(2*a)
print('x1=',x1,', x2=',x2)
print('p(x1)=',a*x1**2+b*x1+c)
print('p(x2)=',a*x2**2+b*x2+c)
```

Fórmula de Baskara

La manera de evitar este tipo de error, modificaremos la fórmula racionalizando el numerador:

Si $b > 0$,

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \frac{-b - \sqrt{b^2 - 4ac}}{-b - \sqrt{b^2 - 4ac}} = \frac{-2c}{b + \sqrt{b^2 - 4ac}},$$

y si $b < 0$,

$$x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a} \frac{-b + \sqrt{b^2 - 4ac}}{-b + \sqrt{b^2 - 4ac}} = \frac{-2c}{b - \sqrt{b^2 - 4ac}}.$$

Tomando esto en cuenta, podemos modificar el programa que escribimos para calcular éstas raíces y hacer una nueva versión que se comporta mejor.

Fórmula de Baskara

```
def raizCuad(a,b,c):
  D=b**2-4*a*c
  if D>0:
    if b>=0:
      x1=-2*c/(b+m.sqrt(D))
      x2=(-b-m.sqrt(D))/(2*a)
    else:
      x1=(-b-m.sqrt(D))/(2*a)
      x2=-2*c/(b-m.sqrt(D))
    return [x1,x2]
  elif D==0:
    x=-b/(2*a)
    return x
  else:
    return "no existen raíces reales"

a=1; b=62.1; c=1
xnuevo=raizCuad(a,b,c)
print(xnuevo)
print('p(x1)=',a*xnuevo[0]**2+b*xnuevo[0]+c)
print('p(x2)=',a*xnuevo[1]**2+b*xnuevo[1]+c)
```

Restando números muy parecidos

En general, se recomienda reescribir las fórmulas donde pueda existir una diferencia cercana a cero.

▷ e.g.: Para $b > 0$,

$$\log(b) - \log(b + \varepsilon) = \log\left(\frac{b}{b + \varepsilon}\right).$$

Calculando el número e

Un problema típico que surge cuando usamos series de Taylor para aproximar funciones es el de sumar a valores cada vez más grandes incrementos cada vez más chicos.

▷ e.g.: para un valor grande de n sabemos que

$$e \approx \sum_{i=0}^{n} \frac{1}{i!} = \sum_{j=0}^{n} \frac{1}{(n-j)!}.$$

Veamos que pasa cuando realizamos estas dos sumas en Python...

Solución: Realizar sumas de menor a mayor para un listado de números positivos.

Representación de punto flotante

Aritmética de punto flotante

Como ya lo mencionamos anteriormente, los números se almacenan en la memoria de la computadora como una cadena de ceros y unos. El sistema más usado es la **aritmética de punto flotante de doble precisión**.
En la aritmética de punto flotante de doble precisión, cada número se representa utilizando 64 bits

$$\underbrace{0}_{s}\ \underbrace{10000000011}_{e}\ \underbrace{1011100100010000000000000000000000000000000000000000}_{m}$$

partido en 3 binarios:

- el signo s de 1 dígito,
- el exponente e, un entero con representación binaria de 11 dígitos excluyendo 00000000000 y 11111111111 y con un corrimiento que permita representar exponentes negativos, y
- la mantisa m, real menor a uno con representación fraccionaria binaria de 52 dígitos.

Aritmética de punto flotante: el número más grande

Ej.: ¿Cuál es el número más grande y más chico que se puede representar con una aritmética finita de base 2 y 32 bits?

Para la conversión de binarios a decimales es bueno recordar que

$$\sum_{i=0}^{n} r^i = \frac{1-r^{n+1}}{1-r}, \quad \sum_{i=1}^{n} r^i = \frac{r(1-r^n)}{1-r}.$$

Como el exponente e cumple

$$0 < c < \sum_{i=0}^{10} 2^i = 2^{11} - 1 = 2047,$$

el número más chico: machine ϵ

¿Qué podemos decir sobre la separación entre puntos de esa nube? Para entender esto deberemos introducir un valor conocido como épsilon de la máquina, que suele denotarse con ϵ. Formalmente, ϵ es el número positivo más pequeño de la aritmética que satisface

$$1 + \epsilon > 1.$$

De esto se deduce que no existe ningún representante de la aritmética entre 1 y $1 + \epsilon$. En esta aritmética se sabe que $\epsilon = 2^{-52} \approx 2.2204 \times 10^{-16}$.

Ej.: Si tengo un número x que se puede representar de manera exacta en una aritmética finita, el siguiente número que puedo representar es $x + \epsilon$ ¿Verdaro o Falso?

floats en la recta real

Si graficamos sobre la recta real los números representables en la aritmética de punto flotante de doble precisión, obtenemos una especie de "nube" de puntos. Para tener una mejor descripción de esta nube, veamos algunas propiedades:

- el número positivo más pequeño se obtiene tomando $s = 0$, $e = 1$, $m = 0$ y es
$$2^{-1022} \approx 2.2251 \times 10^{-308},$$
- el número positivo más grande se obtiene tomando $s = 0$, $e = 2046$, $m = 1 - 2^{-52}$ y es
$$2^{1023}(1 + 1 - 2^{-52}) \approx 1.7977 \times 10^{308},$$
- es simétrica en relación al cero.

Aritmética de punto flotante

Para **poder representar números con exponente negativo**, a e se le resta 1023, obteniendo exponentes enteros en el intervalo $(-1023, 1024)$. Además, para tener siempre una mantisa con parte entera igual a uno, sumaremos 1 a f. Con esto, los números no nulos representables en esta aritmética son

$$x = (-1)^s 2^{e-1023}(1+m).$$

Se define al cero tomando $e = 0$ y $m = 0$, obteniendo dos representaciones: cuando $s = 0$ y cuando $s = 1$.

Aritmética de punto flotante: ejemplo

Escribamos el número real cuya representación en 64 bits es

$$\underbrace{0}_{s}\ \underbrace{10000000011}_{c}\ \underbrace{1011100100010000000000000000000000000000000000000000}_{f}$$

Por un lado tenemos el exponente

$$\begin{aligned}(10000000011)_2 &= 1\cdot 2^{10}+0\cdot 2^9+0\cdot 2^8+0\cdot 2^7+0\cdot 2^6+0\cdot 2^5\\ &\quad +0\cdot 2^4+0\cdot 2^3+0\cdot 2^2+1\cdot 2^1+1\cdot 2^0\\ &= 1024+2+1\end{aligned}$$

Luego,

$$e = 1027$$

Tener en cuenta que el exponente está "corrido" en 1023 para permitir exponentes negativos.

Aritmética de punto flotante: ejemplo

Por otro lado, tenemos la mantisa:

$$\begin{aligned} m &= 1\cdot\left(\frac{1}{2}\right)^1 + 0\cdot\left(\frac{1}{2}\right)^2 + 1\cdot\left(\frac{1}{2}\right)^3 + 1\cdot\left(\frac{1}{2}\right)^4 + 1\cdot\left(\frac{1}{2}\right)^5 + 0\cdot\left(\frac{1}{2}\right)^6 \\ &\quad +0\cdot\left(\frac{1}{2}\right)^7 + 1\cdot\left(\frac{1}{2}\right)^8 + \ldots + 1\cdot\left(\frac{1}{2}\right)^{12} + \ldots + 0\cdot\left(\frac{1}{2}\right)^{52} \\ &= \frac{1}{2} + \frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \frac{1}{256} + \frac{1}{4096} \end{aligned}$$

De esto obtenemos que el número real es

$$\begin{aligned} x &= (-1)^s(1+m)2^{e-1023} \\ x &= \left(1 + \frac{1}{2} + \frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \frac{1}{256} + \frac{1}{4096}\right) 2^4 = 27.56640625 \end{aligned}$$

Undeflow / overflow

Cuando el resultado de un cálculo es un número de magnitud menor a la representable, se está en presencia de un **underflow**.

En general el valor se suele sustituir por cero y los cálculos continúan.
Cuando el resultado es un número de magnitud mayor a la representable, se está en presencia de un **overflow**. Generalmente los cálculos se detienen.

Errores de representación

El representante en punto flotante de un número x, denotado por $\mathrm{fl}(x)$, se define como aquél que minimiza la distancia entre x y los puntos de la aritmética.
En el caso de existir dos posibilidades (cuando x es justo el punto medio entre dos representantes) se toma el representante de mayor magnitud. Este método para elección de representante se conoce como redondeo y es el más usado.

▷ e.g.: si tenemos tres números positivos consecutivos en la aritmética $a < b < c$, tendremos

$$\mathrm{fl}(x) = b \quad \text{si } x \in \left[\frac{a+b}{2}, \frac{b+c}{2} \right).$$

Errores de representación: truncamiento y redondeo

Por ejemplo, si tenemos tres números positivos consecutivos en la aritmética $a < b < c$, tendremos

$$\mathrm{fl}(x) = b \quad \text{si } x \in \left[\frac{a+b}{2} , \frac{b+c}{2} \right) .$$

Otro método existente para elegir un representante, conocido como truncamiento asigna siempre el representante de menor magnitud. O sea, dados dos números positivos en la aritmética $a < b$, tendremos

$$\mathrm{fl}(x) = a \quad \text{si } x \in [a , b) .$$

Cada vez que un número x no pertenezca a la aritmética, estaremos generando un error de redondeo. Llamaremos

- error absoluto al valor $|x - \mathrm{fl}(x)|$
- error relativo al valor $\dfrac{|x - \mathrm{fl}(x)|}{|x|}$

Operaciones de punto flotante

Por otro lado, las operaciones se van ejecutando de izquierda a derecha y el resultado de cada operación es un elemento de la aritmética. Con esto, la operación

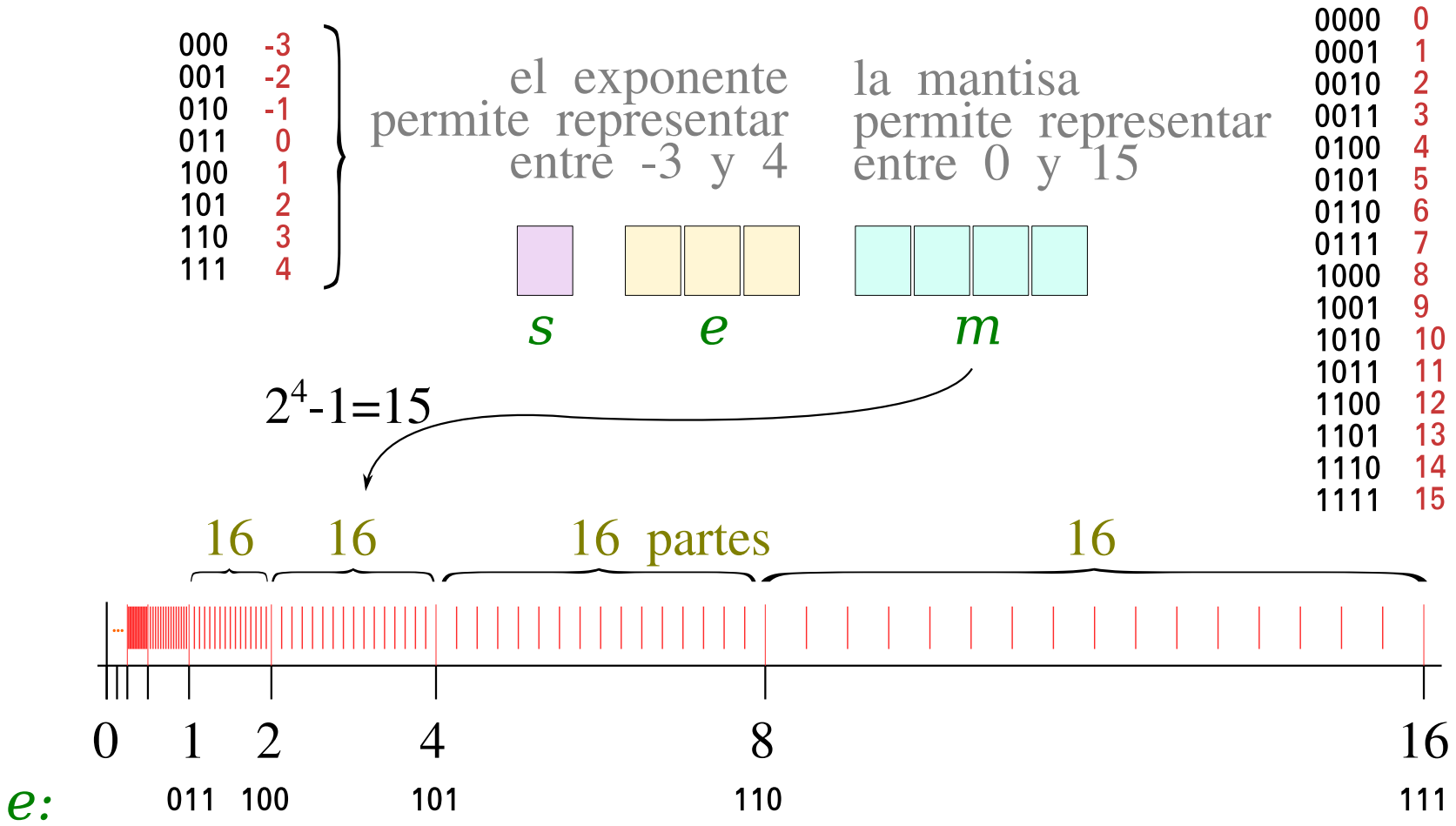
$$x + y + z$$

dará como resultado

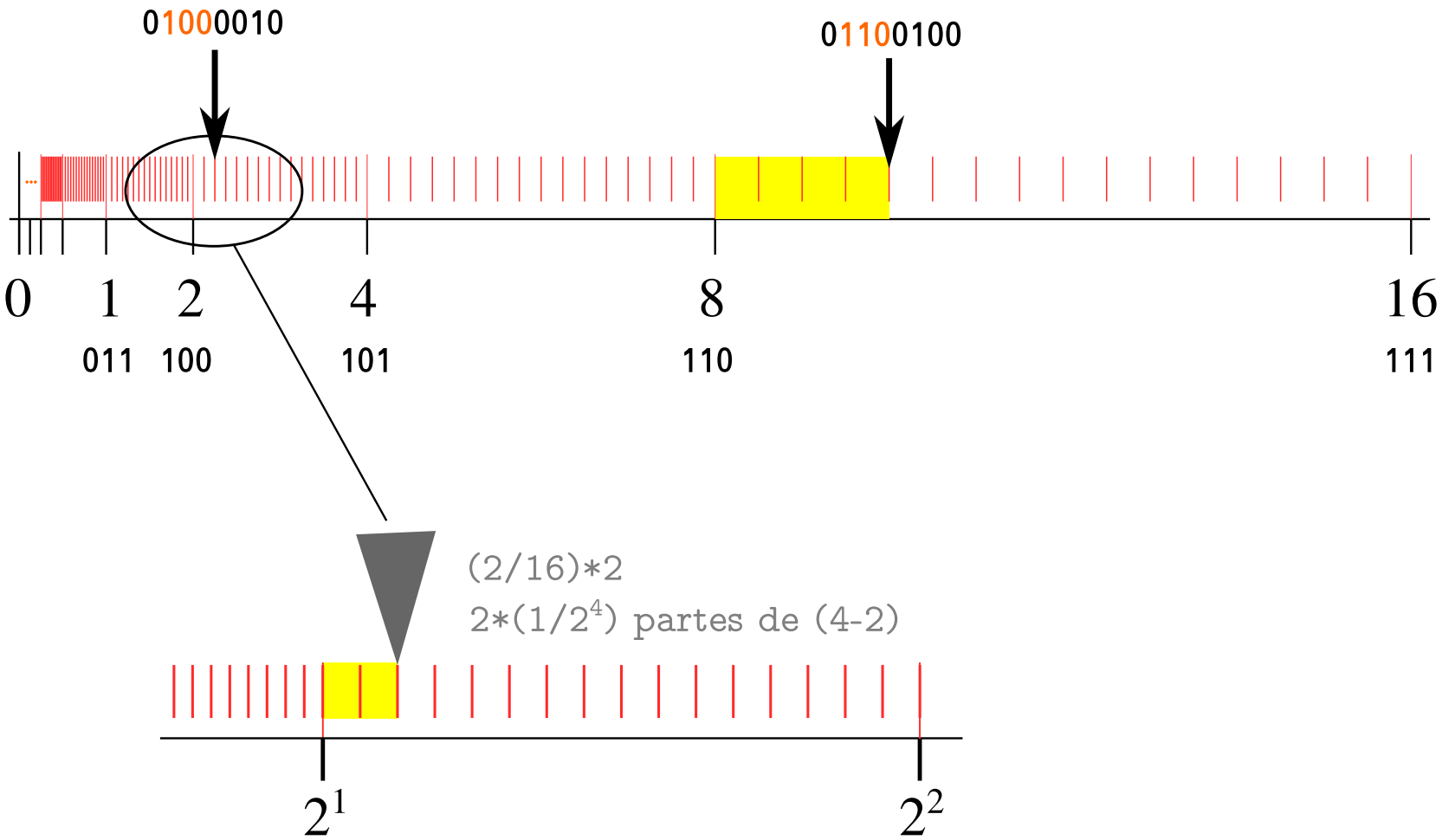
$$\text{fl}\left(\text{fl}\left(\text{fl}(x) + \text{fl}(y)\right) + \text{fl}(z)\right).$$

Esto hace que los resultados a veces no sean los esperaados...

Aritmética de punto flotante de 8 bits



▷ *e.g.: Suma en 8 bits*



▷ *e.g.: Suma en 8 bits*

Queremos sumar los números representados en 8 bits por 01000010 y por 01100100.

En el primer caso, tenemos s=0, e=4, m=2:

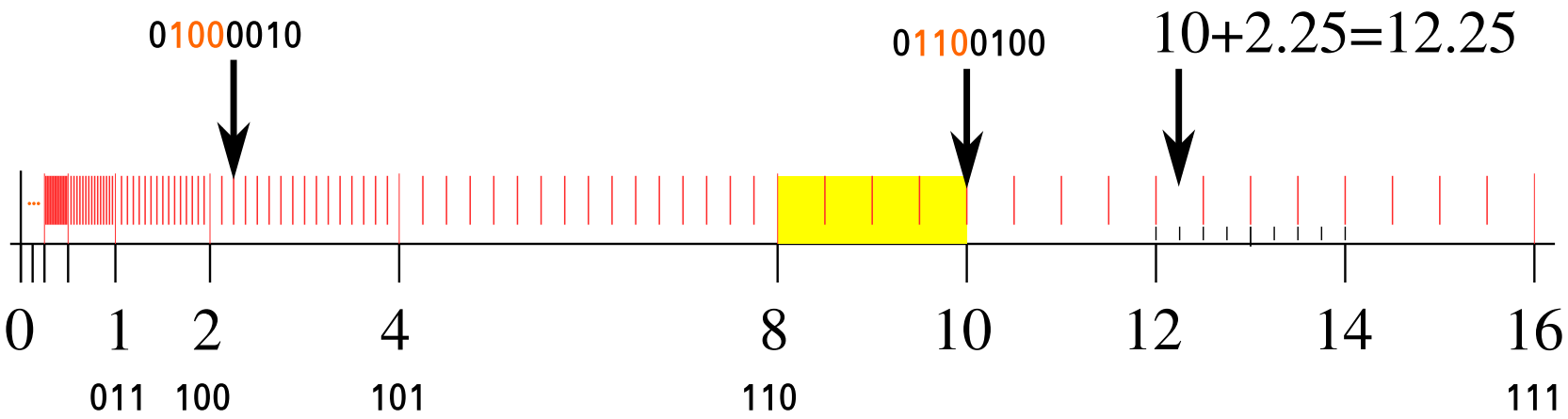
$$\begin{aligned}(01000010)_2 &= (-1)^0 * (1 + 2^{-3}) * 2^{4-3} \\ &= (1 + 1/8) * 2 \\ &= 2.25\end{aligned}$$

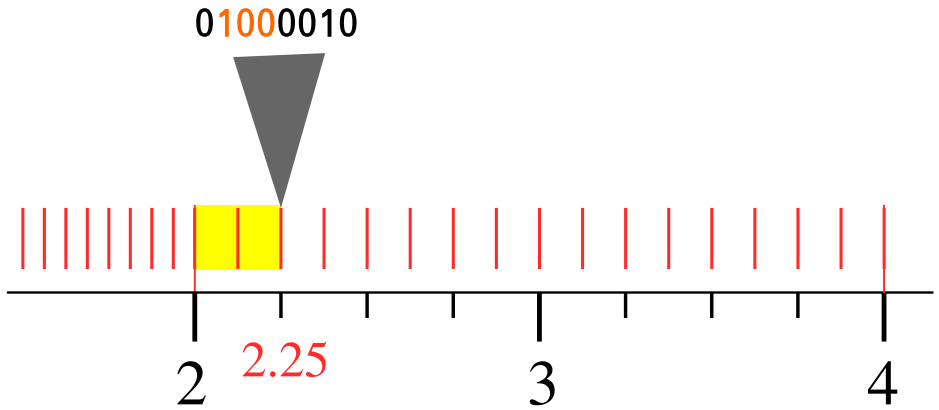
En el otro caso, s=0, e=6, m=4:

$$\begin{aligned}(01100100)_2 &= (-1)^0 * (1 + 2^{-2}) * 2^{6-3} \\ &= (1 + 1/4) * 2^3 \\ &= 10\end{aligned}$$

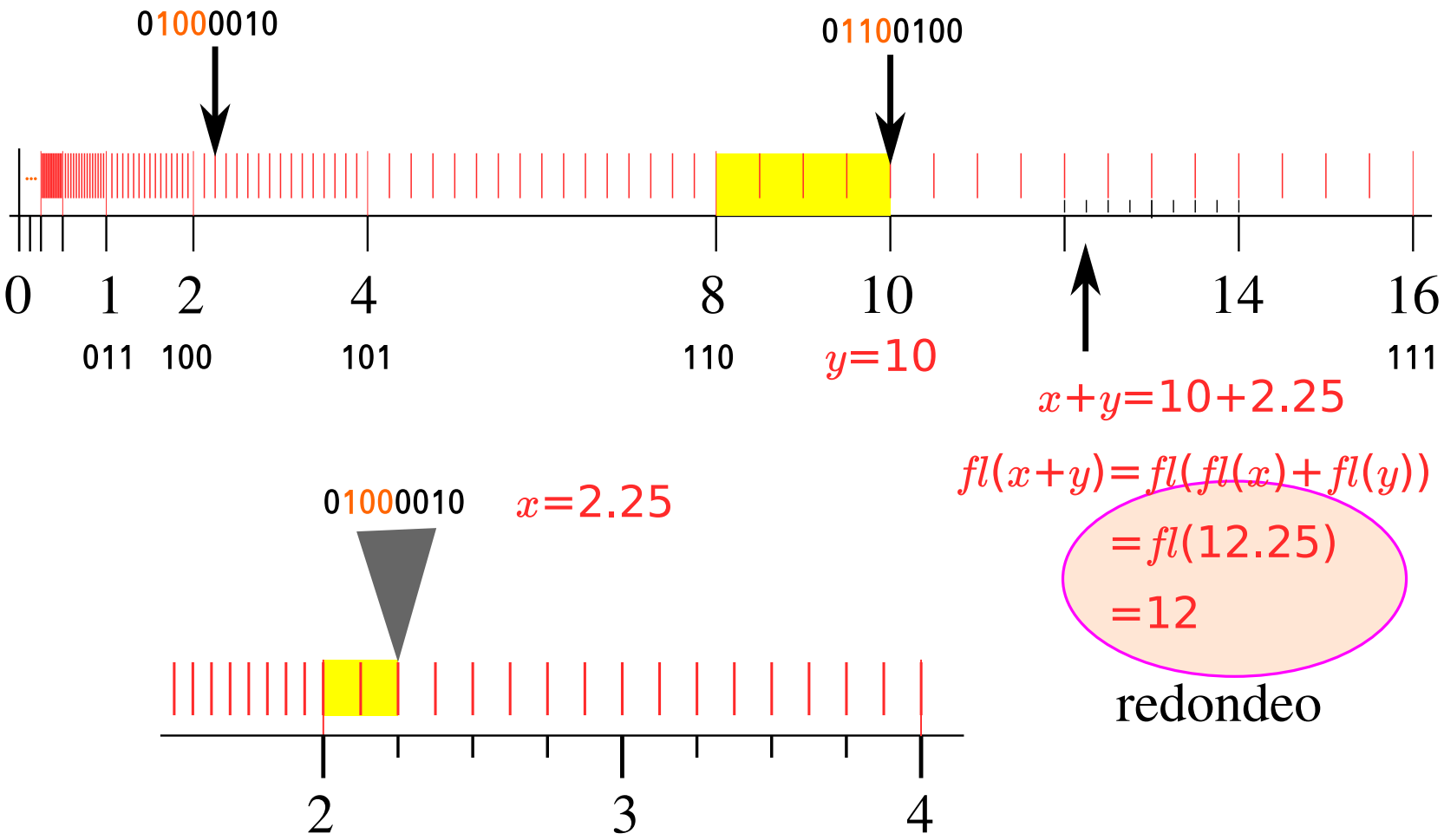
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100

▷ *e.g.: Suma en 8 bits*

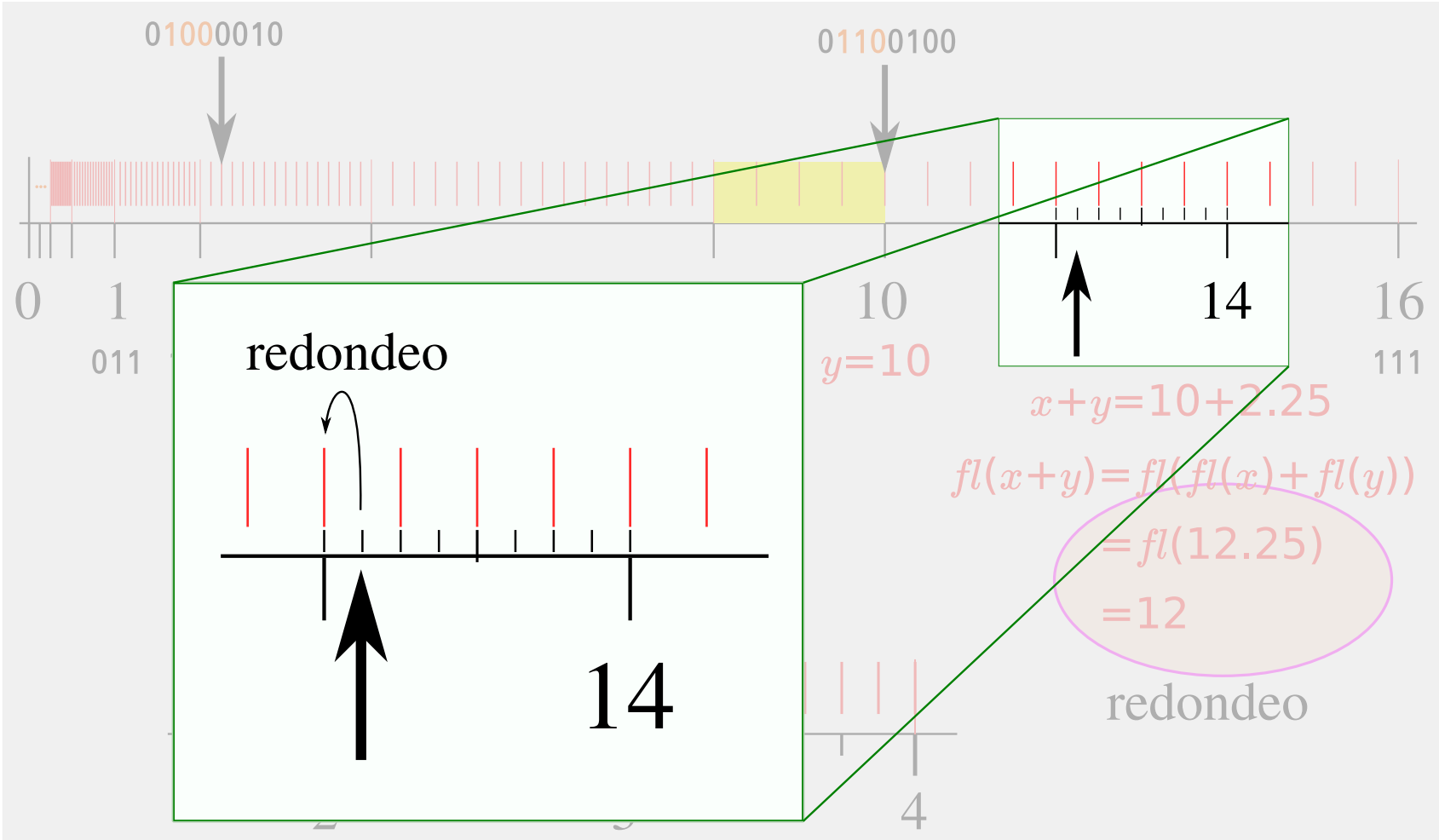




▷ *e.g.: Suma en 8 bits*



▷ *e.g.: Suma en 8 bits*



▷ *e.g.: Suma en 8 bits*

$$
\begin{aligned}
10 &= (01100100)_2 \\
&= (-1)^0 \cdot (1 + 2^{-2}) \cdot 2^{6-3} \\
&= (1 + 1/4) \cdot 2^3 \\
&= (1 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2}) \cdot 2^3 \\
&= (1.01)_2 \cdot 2^3
\end{aligned}
$$

$$
\begin{aligned}
2.25 &= (01000010)_2 \\
&= (-1)^0 \cdot (1 + 2^{-3}) \cdot 2^{4-3} \\
&= (1 + 1/8) \cdot 2 \\
&= (1 + 0 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-2}) \cdot 2^3 \\
&= (1.001)_2 \cdot 2 \\
&= (0.01001)_2 \cdot 2^3
\end{aligned}
$$

▷ *e.g.: Suma en 8 bits*

$$10 = (01100100)_2 = (1.01)_2 \cdot 2^3$$

$$2.25 = (01000010)_2 = (0.01001)_2 \cdot 2^3$$

$$\begin{array}{rccccccc} & 1 & . & 0 & 1 & 0 & 0 & \times 2^3 \\ + & 0 & . & 0 & 1 & 0 & 1 & \times 2^3 \\ \hline & 1 & . & 1 & 0 & 0 & 0 & \times 2^3 \end{array}$$

En efecto, $(1.10001)_2 \times 2^3 = (1 + 1/2 + 1/32) \times 2^3 = 12.25$
Pero no se puede representar ese número en base 8, por lo que lo tenemos que "redondear", y queda:

$$(1.1000)_2 \times 2^3 = (1 + 1/2) \times 2^3 = 12 \implies fl(10 + 2.25) = (1.1000)_2 \times 2^3$$

▷ *e.g.: Suma en 8 bits*

$$(1.1000)_2 \times 2^3 = (1 + 1/2) \times 2^3 = 12 \implies fl(10 + 2.25) = (1.1000)_2 \times 2^3$$

que tiene una representación de punto flotante en 8 bits de: `01101000`

▷ *e.g.:* $\sqrt{3}$

Claramente $\sqrt{3}$ no tiene una cantidad finita de dígitos decimales y por lo tanto no puede representarse en esta aritmética. En este caso Python utiliza el representante más próximo dentro de la aritmética.

```
from math import sqrt
a = sqrt(3)
print(a)
```

El cuadrado de este representante no es igual a 3. Por otro lado, como 3^2 si pertenece a la aritmética, tenemos que:

```
sqrt(3**2)==3
```

Operaciones de punto flotante, ▷ e.g.: $1 + 2^{-53} + 2^{-53}$

Al ejecutar $1 + 2^{-53} + 2^{-53}$ se obtiene

$$\text{fl}\left(\text{fl}\left(\text{fl}(1) + \text{fl}(2^{-53})\right) + \text{fl}(2^{-53})\right) = \text{fl}\left(\text{fl}\left(1 + 2^{-53}\right) + 2^{-53}\right) = \text{fl}\left(1 + 2^{-53}\right) =$$

ya que 2^{-53} es menor que el épsilon de la máquina.
Al ejecutar $2^{-53} + 2^{-53} + 1$ se obtiene

$$\text{fl}\left(\text{fl}\left(\text{fl}(2^{-53}) + \text{fl}(2^{-53})\right) + \text{fl}(1)\right) = \text{fl}\left(\text{fl}\left(2^{-53} + 2^{-53}\right) + 1\right) = \text{fl}\left(2^{-52} + 1\right) =$$

ya que 2^{-52} es el épsilon de la máquina.

notación de orden asimptótico

Vimos que aparecen muchos problemas numéricos y algunas estrategias para mitigarlos. Pero, ¿cómo sabemos si estamos cometiendo errores?

Necesitamos una métrica que nos permita tener una idea o cota del error que cometemos.

Deberemos utilizar métodos que generen sucesiones que irán aproximándose a la solución del problema. Éstos se conocen como **métodos de aproximación**.

En general, para los problemas en donde la solución no está dada por una fórmula, la estrategia es utilizar un método que va calculando aproximaciones cada vez mejores.

Aproximaciones

Uno de los aspectos a tener en cuenta en estos métodos es la **estabilidad**.

Diremos que un método es:

- estable si pequeñas variaciones en los datos iniciales producen pequeñas variaciones en los resultados,
- inestable si pequeñas variaciones en los datos iniciales producen variaciones grandes en los resultados, y
- condicionalmente estable si son estables solo para algunos datos iniciales.

En la mayoría de los casos se puede estimar el error cometido en la iteración n, digamos E_n. A medida que vayamos aproximándonos a la solución, E_n debería tender a cero.

Velocidad de convergencia

Como los procedimientos tendrán una cantidad finita de pasos, nos gustaría que esta sucesión converja lo más rápidamente posible. Para estimar la **velocidad de convergencia** usaremos la notación O y o.

Def./

Dada una sucesión $t_n \to 0^+$ y una función $F : (0, \infty) \mapsto \mathbb{R}$, denotaremos:

- $F(t_n) = O(t_n)$ si existe $c > 0$ tal que $\frac{|F(t_n)|}{t_n} < c$, para n suficientemente grande, y diremos que $F(t_n)$ converge a cero con orden (o con velocidad de convergencia) $O(t_n)$.
- $F(t_n) = o(t_n)$ si para todo $\varepsilon > 0$ vale que $\frac{|F(t_n)|}{t_n} < \varepsilon$, para n suficientemente grande, y diremos que $F(t_n)$ converge a cero con orden $o(t_n)$

Velocidad de convergencia

De manera equivalente,

- $F(t_n) = O(t_n)$ si $\limsup_{n\to\infty} \dfrac{|F(t_n)|}{t_n} < \infty$,
- $F(t_n) = o(t_n)$ si $\lim_{n\to\infty} \dfrac{|F(t_n)|}{t_n} = 0$.

Cuando analizamos la velocidad de convergencia del error E_n decimos que converge:

- linealmente, si $E_{n+1} = O(E_n)$ con $c < 1$,
- superlinealmente, si $E_{n+1} = o(E_n)$,
- cuadráticamente, si $E_{n+1} = O(E_n^2)$.

▷ *e.g.: media aritmética*

Def./

Media aritmética-geométrica (agm): dados dos números reales positivos x e y, se generan sucesiones $\{a_n\}$ y $\{b_n\}$ tales que $a_0 = x$, $b_0 = y$,

$$a_{n+1} = \frac{a_n + b_n}{2}, \quad b_{n+1} = \sqrt{a_n b_n}.$$

Se sabe que $\{a_n\}$ es decreciente, $\{b_n\}$ es creciente, $b_n \leq \text{agm}(x,y) \leq a_n$ y ambas convergen a

$$\text{agm}(x,y) = \frac{\pi}{2}\left(\int_0^{\pi/2} \frac{d\theta}{\sqrt{x^2\cos^2(\theta) + y^2\text{sen}^2(\theta)}}\right)^{-1}.$$

▷ *e.g.:*

Suponga que se tienen las sucesiones

$$x_n = \frac{n+1}{n^2}, \quad y_n = \frac{n+3}{n^3},$$

para $n \geq 1$ con $x_0 = y_0 = 5$.

```
N=10
x=[5]; y=[5]
for n in range(1,N):
  x.append((n+1)/n**2)
  y.append((n+3)/n**3)
print(x)
print(y)
plt.figure()
plt.plot(x)
plt.plot(y)
plt.title("{xn}, {yn}");
```

▷ *e.g.: Velocidad de convergencia*

Si analizamos la velocidad de convergencia notaremos que

$$\frac{x_n}{1/n} = 1 + \frac{1}{n} \to 1, \quad \frac{x_n}{1/n^2} = n + 1 \to \infty,$$

por lo tanto $x_n = O(\frac{1}{n})$, y como

$$\frac{y_n}{1/n} = \frac{n+3}{n^2} \to 0, \quad \frac{y_n}{1/n^2} = 1 + \frac{3}{n} \to 1,$$

entonces $y_n = o(\frac{1}{n})$ y $y_n = O(\frac{1}{n^2})$.

▷ *e.g.: Fibonacci*

En algunos casos pasa que para algunos valores de p la sucesión converge y para otros diverge.
Considere la sucesión de Fibonacci:

$$F_0 = 1, \quad F_1 = 1, \quad F_{n+2} = F_n + F_{n+1}.$$

Para analizar la velocidad de convergencia definimos

$$x_n = \frac{F_{n+1}}{F_n}.$$

Si existiera $x = \lim_{n\to\infty} x_n$, como

$$x_n = \frac{F_{n-1} + F_n}{F_n} = \frac{F_{n-1}}{F_n} + 1 = \frac{1}{x_{n-1}} + 1,$$

tomando límite obtendríamos que $x = 1/x + 1$ y por lo tanto cumple la ecuación $x^2 - x - 1 = 0$. O sea,

$$x = \frac{1 + \sqrt{5}}{2}.$$

Qué aprendimos

- ¡el orden en el que hacemos las operaciones importa! (en computación, aunque no analíticamente)
- ¡cuidado con las sumas de muchos términos y las restas de números parecidos!
- cuando hay que calcular algo que no se puede hacer de manera exacta, reemplazamos la función por algo más simple que la aproxima, y hacemos el cálculo exacto en la aproximación simple.

Aproximaciones de π

Aproximación de Arquímedes

Esta aproximación geométrica se basa en calcular el perímetro de polígonos inscriptos y circunscriptos. Más específicamente, se comienza con el perímetro de un hexágono regular inscripto y uno circunscripto a la circunferencia de radio $r = \frac{1}{2}$. La estrategia consiste en ir duplicando la cantidad de lados en cada iteración. Se puede ver que los perímetros de estos hexágonos van a converger a

$$2\pi r = \pi.$$

Aproximación de Arquímedes

Suponga que la circunferencia tiene radio r, α es el ángulo desde el eje x, ℓ_0 es el lado inicial del poliedro inscripto y L_0 el lado inicial del poliedro circunscripto. Por lo tanto

$$\frac{\ell_0}{2} = r\,\mathrm{sen}(\alpha), \quad \frac{L_0}{2} = r\tan(\alpha).$$

Cuando dividamos el ángulo en 2, los nuevos lados cumplirán

$$\frac{\ell_1}{2} = r\,\mathrm{sen}\left(\frac{\alpha}{2}\right), \quad \frac{L_1}{2} = r\tan\left(\frac{\alpha}{2}\right).$$

Para obtener una fórmula iterativa, escribamos las funciones aplicadas a $\alpha/2$ en términos de las aplicadas en α. Recuerde que
$\cos(2x) = \cos^2(x) - \mathrm{sen}^2(x) = 1 - 2\,\mathrm{sen}^2(x)$. Por lo tanto

$$\mathrm{sen}\left(\frac{\alpha}{2}\right) = \sqrt{\frac{1-\cos(\alpha)}{2}} = \sqrt{\frac{1-\sqrt{1-\mathrm{sen}^2(\alpha)}}{2}}.$$

Aproximación de Arquímedes

Como $\operatorname{sen}(2x) = 2\operatorname{sen}(x)\cos(x)$ entonces

$$\tan(2x) = \frac{\operatorname{sen}(2x)}{\cos(2x)} = \frac{2\operatorname{sen}(x)\cos(x)}{\cos^2(x) - \operatorname{sen}^2(x)} = \frac{2\tan(x)}{1 - \tan^2(x)},$$

Que es equivalente a $0 = \tan(2x)\tan^2(x) + 2\tan(x) - \tan(2x)$ y como $\tan(x) > 0$ tenemos que

$$\tan(x) = \frac{-1 + \sqrt{1 + \tan^2(2x)}}{\tan(2x)},$$

o sea

$$\tan\left(\frac{\alpha}{2}\right) = \frac{-1 + \sqrt{1 + \tan^2(\alpha)}}{\tan(\alpha)}.$$

Aproximación de Arquímedes

Ahora, si la circunferencia tiene radio $r = \frac{1}{2}$ y comenzamos con hexágonos entonces $s_0 = \text{sen}(\alpha) = \frac{1}{2}$ y $t_0 = \tan(\alpha) = \frac{1}{\sqrt{3}}$. Los perímetros del hexágono inscripto y circunscripto son

$$P_0 = 6\ell_0 = 6s_0, \quad Q_0 = 6L_0 = 6t_0.$$

Al duplicar la cantidad de lados tendremos perímetros

$$P_1 = 2 \cdot 6\ell_1 = 2 \cdot 6s_1, \quad Q_1 = 2 \cdot 6L_1 = 2 \cdot 6t_1,$$

donde

$$s_1 = \sqrt{\frac{1 - \sqrt{1 - s_0^2}}{2}}, \quad t_1 = \frac{\sqrt{1 + t_0^2} - 1}{t_0}.$$

Aproximación de Arquímedes

Si contunuamos con este procedimiento obtendremos que

$$P_n = 2^n \cdot 6s_n, \quad Q_n = 2^n \cdot 6t_n,$$

donde

$$s_n = \sqrt{\frac{1 - \sqrt{1 - s_{n-1}^2}}{2}}, \quad t_n = \frac{\sqrt{1 + t_{n-1}^2} - 1}{t_{n-1}}.$$

Se puede demostrar que $P_n < \pi < Q_n$ y ambas sucesiones convergen a π.
Veamos esto primero en
[GeoGebra](https://www.geogebra.org/m/mjpzurkk) y luego en Python.

Aproximación de Wallis

John Wallis (1616 - 1703) escribió π como un producto infinito de racionales.

$$\pi = 2 \prod_{n=1}^{\infty} \frac{4n^2}{4n^2 - 1}.$$

Aproximación de Euler

Leonhard Euler (1707 - 1783). Obtuvo la aproximación estudiando las sumas invertidas de las raíces de polinomios.

$$\pi = \sqrt{6 \sum_{k=1}^{\infty} \frac{1}{k^2}}$$

Algoritmo de Brent-Salamin

Está relacionado con el algoritmo de la media aritmética-geométrica visto la clase pasada. Publicada en 1976 de manera independiente por Eugene Salamin y luego por Richard Brent. El procedimiento tiene valores iniciales

$$a_0 = 1, \quad b_0 = \frac{1}{\sqrt{2}}, \quad u_0 = \frac{1}{4}, \quad v_0 = 1,$$

y generar las sucesiones

$$\begin{aligned} a_{n+1} &= \frac{a_n + b_n}{2}, \\ b_{n+1} &= \sqrt{a_n b_n}, \\ u_{n+1} &= u_n - v_n(a_n - a_{n+1})^2, \\ v_{n+1} &= 2v_n. \end{aligned}$$

Obteniendo que

$$\pi \approx \frac{(a_{n+1} + b_{n+1})^2}{4u_{n+1}}.$$