Protocolo de Construcción

Mono y Juli

16 de diciembre de 2020

Error absoluto y relativo

En este recurso nos encontramos con un Chad que debe golpear en la base de la columna para elevar la bolita roja a la meta, o lo mas cerca posible.

Lamentablemente, casi nunca llegara exactamente a la meta, habrá un error, y le pedimos a los usuarios que calculen dicho error.

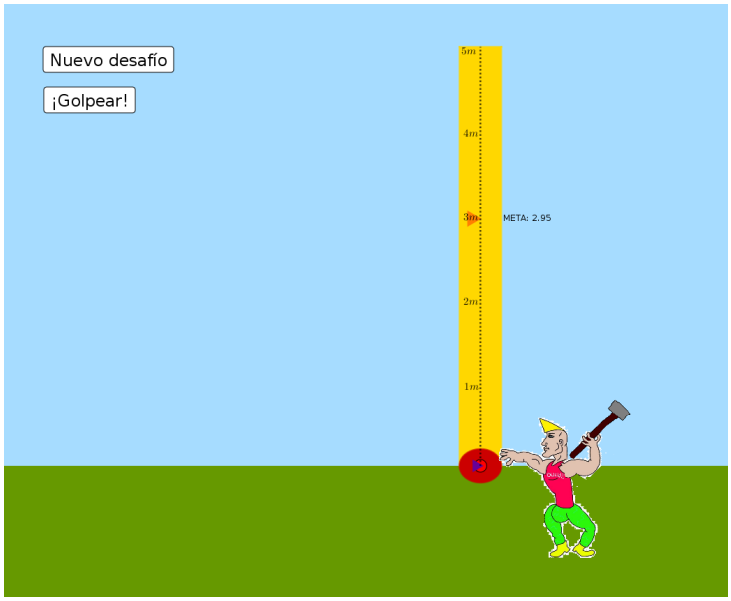


Figura 1: Pantallazo del recurso.

Nuevo desafío

Definimos 2 variables que son centrales: `meta` y `errorabs` (`meta` es el valor al que el musculoso pretende llegar golpeando la bolita, y `errorabs` es el error absoluto que tendra). Al apretar el botón *nuevo desafío* se le asigna un valor aleatorio a ambas variables; para mantener controlados los posible resultados (salvar a los usuarios de tener que escribir 15 decimales) restringimos los posibles valores que pueden tomar las varaiables con una instruccion (ver Figura 2) del tipo:

```
Valor(meta, AleatorioEntre(35,95)/10)
```

que le asigna a `meta` un número aleatorio entero entre 35 y 95, pero al dividirlo por 10 conseguimos un número con una sola cifra decimal entre 3.5 y 9.5.[1]

[1] Si bien la columna en el dibujo tiene 5 metros, en realidad está parametrizada como una recta de 0 a 10, por eso los valores se generan hasta 9.5.

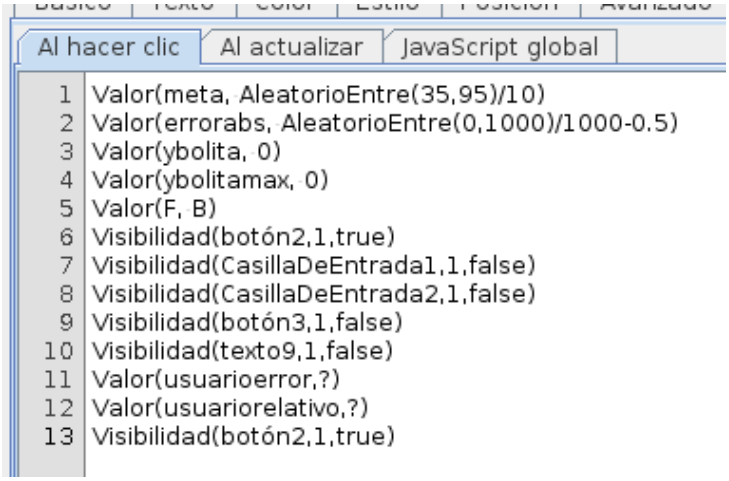
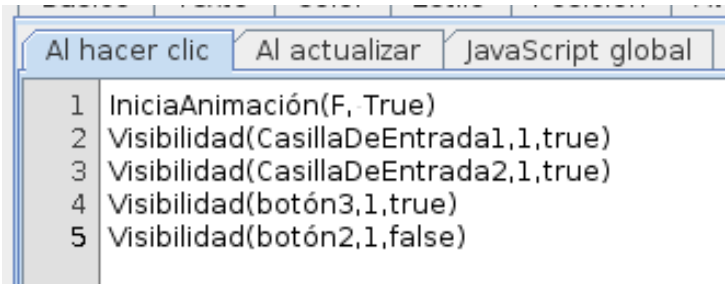


Figura 2: Código en el botón *Nuevo desafío*.

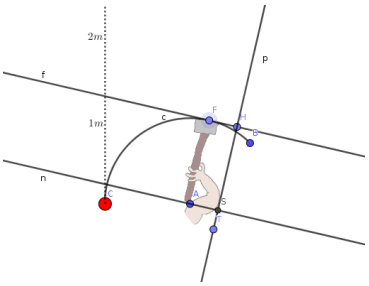
Además de eso, el botón *nuevo desafío* restablece otras condiciones iniciales como colocar la bolita en la base de la columna, ocultar las casillas para colocar una respuesta y comprobarla; y hace visible el botón de *golpear*.

Golpear!

Al hacer click en *golpear* se inicia la animación de F, un punto que se mueve por un arco de circunferencia que describe la trayectoria de la cabeza del martillo, el arco es de centro A, de puntos inicial B y final C. A artir de la recta tangente al arco que pasa por F se define toda una estructura de rectas y puntos que sirve para ajustar el tamaño y la posición de la imagen de forma que parezca que el centro de la rotación es el hombro.



(a) Código en el botón "Golpear".



(b) Estructura para ajustar la imagen del martillo.

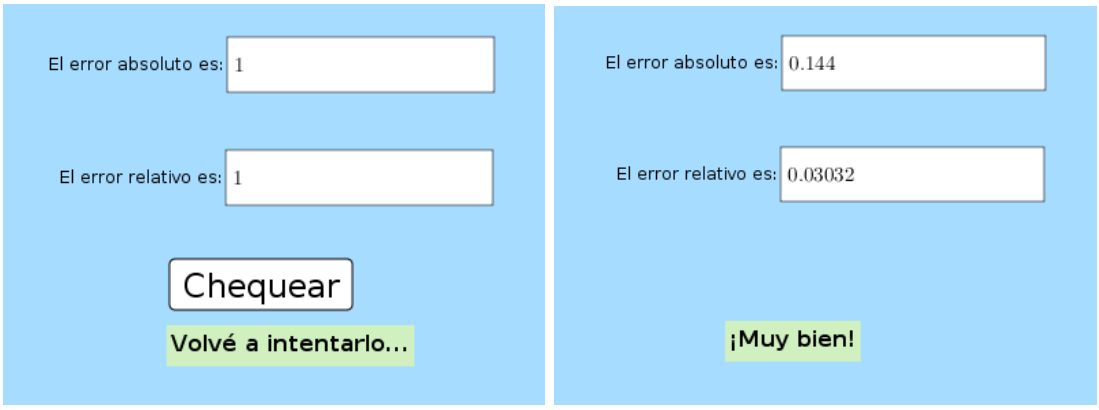
Figura 3: Comandos y estructura detrás de la animación del golpe.

F parte de B y al animarlo recorre todo el arco una sola vez hasta llegar a C, donde una instrucción en la pestaña *al actualizar* del mismo F inicia la animación de otros 2 deslizadores: `ybolita` e `ybolitamax`. Estos son parámetros de la función `altura(t)`, que describe la trayectoria de la bolita; `ybolita` se mueve de forma que `altura(ybolita)` (que describe la coordenada y de la bolita) termine en el suelo y `ybolitamax` se mueve de forma que `altura(ybolitamax)` (que describe la coordenada y del triangulo que marca la altura máxima que ha alcanzado la bolita) termina en el punto máximo de la parábola. Ambos deslizadores son únicamente de ida.

Ejercicio

Golpear también hace visibles las casillas para ingresar una respuesta, donde se esperan los valores del error absoluto y error relativo. Al hacer click en *chequear* se compraran los valores

ingresados con los reales y en base a eso se muestran los textos *muy bien* o *intenta de nuevo*.



(a) Respuesta equivocada. (b) Respuesta correcta.

Figura 4: Comprobación de respuestas.

Método de Bisección

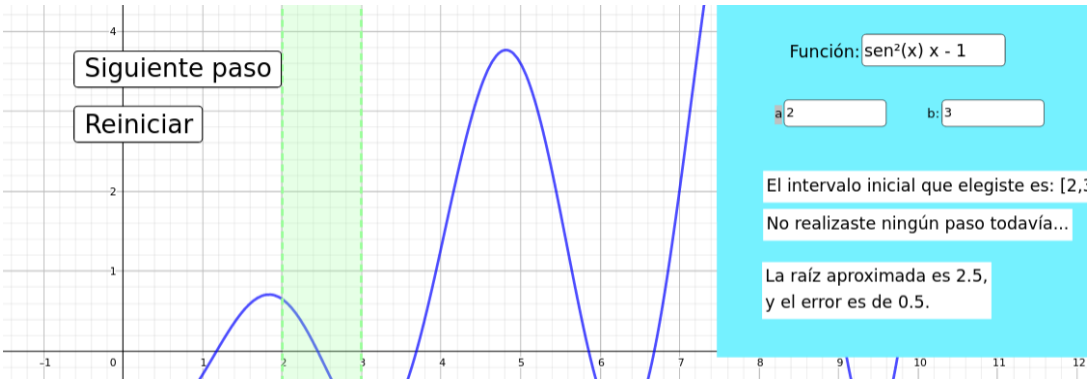


Figura 5: Primer pantallazo del recurso.

En los casilleros de entrada a y b se introducen los límites del intervalo inicial, estos casilleros están ligados a las variables `e` y `d`, es decir: si el usuario modifica los valores en los casilleros, automáticamente cambian los valores en las variables.

Luego, al presionar *reiniciar*, otras 2 variables `a` y `b` toman los valores de `d` y `e`. El área que se grafica (donde esta la raíz) corresponde a la inecuación $\texttt{a} < x < \texttt{b}$, donde `a` y `b` se van actualizando (cambian sus valores) cada vez que se presiona *siguiente*. Si luego el usuario quiere reiniciar el algoritmo, necesitamos almacenar su intervalo inicial en algún lado, no en `a` y `b` porque al presionar *siguiente*, esos valores se actualizaran y perderemos la información, Por eso los almacenamos en `d` y `e`.

Cada vez que se presiona *siguiente paso*, se incrementa un deslizador discreto `n` que cuenta el numero de pasos (con este deslizador luego se arman los textos *realizaste* `n| pasos`) y se actualizan los valores de `a` y `b` según el algoritmo de bisección (ver Figura 6).

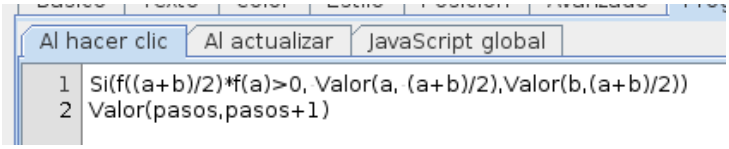


Figura 6: Código del botón "siguiente paso".

Un detalle importante de la construcción de los textos es que se utilizo el comando raíz de Geogebra para comprobar que en el intervalo inicial hubiera una raíz antes de empezar, de no haberla, la variable `root` definida como `root = raiz(funcion, intevalo inicial)` estaría indefinida, entonces podemos mostrar la advertencia "No hay ninguna raiz allíütilizando el comando `EstaDefinido`.[2] Éste comando sin embargo no es perfecto y arroja distintas respuestas dependiendo de la multiplicidad de las raíces, todavía no entendemos del todo como solucionar este error.

Método de Newton

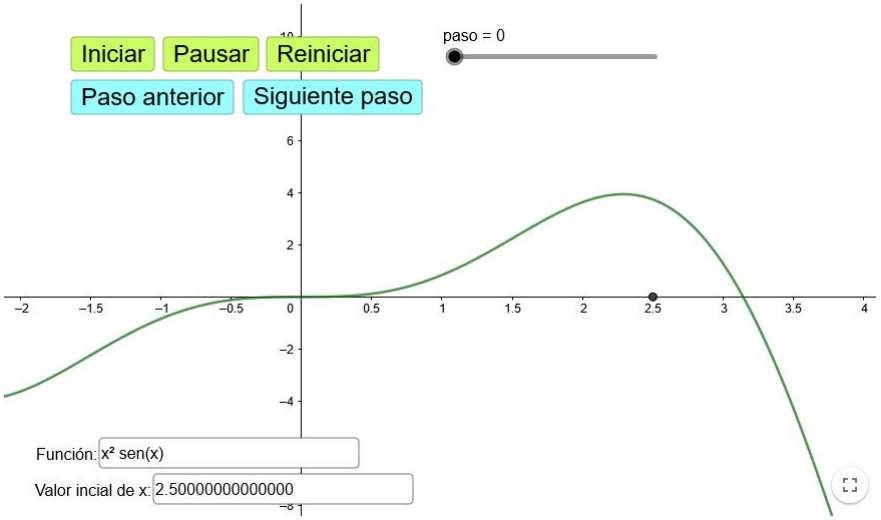


Figura 7: Primer pantallazo del recurso.

Este recurso está orientado a mostrar cómo funciona el algoritmo del método de Newton. Es una buena herramienta para poder visualizar cómo es su convergencia respecto a otros métodos, y qué limitaciones tiene si se elige un punto muy alejado al cero que se busca.

Para armar este recurso, primero se agregan dos casillas de entrada. Una para la función y otra para el valor inicial de x. Estos luego podrán cambiarse a conveniencia del usuario.

Para poder armar el recurso propusimos una función y un valor inicial arbitrarios.

Hacemos un punto I en $(x, 0)$ y otro A en la función evaluada en el valor de x propuesto. Unimos ambos con un segmento punteado color rojo para indicar que una vez elegido el valor de x inicial, debemos centrarnos en el segundo punto.

Luego, se crea la tangente a la función en $x =$ coordenada x del punto A, que se obtiene con `x(A)`. También decidimos ponerle un color que la distinga del segmento vertical trazado anteriormente, y hacer la línea punteada. Con el comando `Raíz` encontramos la intersección de esta recta tangente, con el eje x y creamos un nuevo punto allí para poder visualizarlo. Con este nuevo valor de x, repetimos todos estas construcciones hasta llegar a una cantidad de pasos acorde a la precisión de la aproximación que busquemos obtener.

[2] Gracias Sol.

Nosotros decidimos repetir 2 pasos y quedarnos con la tercera raíz hallada como aproximación al cero de la función. Consideramos que el método converge muy rápidamente como para que sea razonable programar muchos más pasos, y de elegir un punto inicial cercano al cero buscado, se obtendrán muy buenas aproximaciones.

Para poder mostrar el cero aproximado, creamos un texto que tiene incorporado como objeto al último punto que se obtuvo de la raíz de la tangente a la función con el eje x. Para incorporarlo, al escribir el texto seleccionamos el punto en la pestaña objeto.

Luego definimos un deslizador al que llamamos *paso*, que va de 0 a 6. En el 0 sólo es visible la función y el punto inicial. Luego iremos mostrando más elementos en el recurso a medida que aumenta el número que indica el deslizador.

1. Se muestra el segundo punto y el segmento rojo que los une.
2. Se muestra el tercer punto y la recta tangente a la función en el segundo punto.
3. Se muestra el cuarto punto y el segundo segmento rojo.
4. Se muestra el quinto punto y la recta tangente a la función en el cuarto punto.
5. Se muestra el sexto punto y el segmento rojo que lo une con el punto anterior.
6. Se muestra el cero aproximado en color verde con su respectivo rótulo, y la recta tangente a la función que conecta el cero aproximado con el último punto que habíamos obtenido.

Todo esto se logra utilizando el comando `Visibilidad(objeto,1,true)` cada vez que deseamos mostrar un nuevo objeto. El 1 corresponde a la vista gráfica que estamos usando.

Se deja visible este deslizador para que el usuario pueda ir avanzando en pasos a su gusto. Sin embargo, la animación tiene más protagonismo.

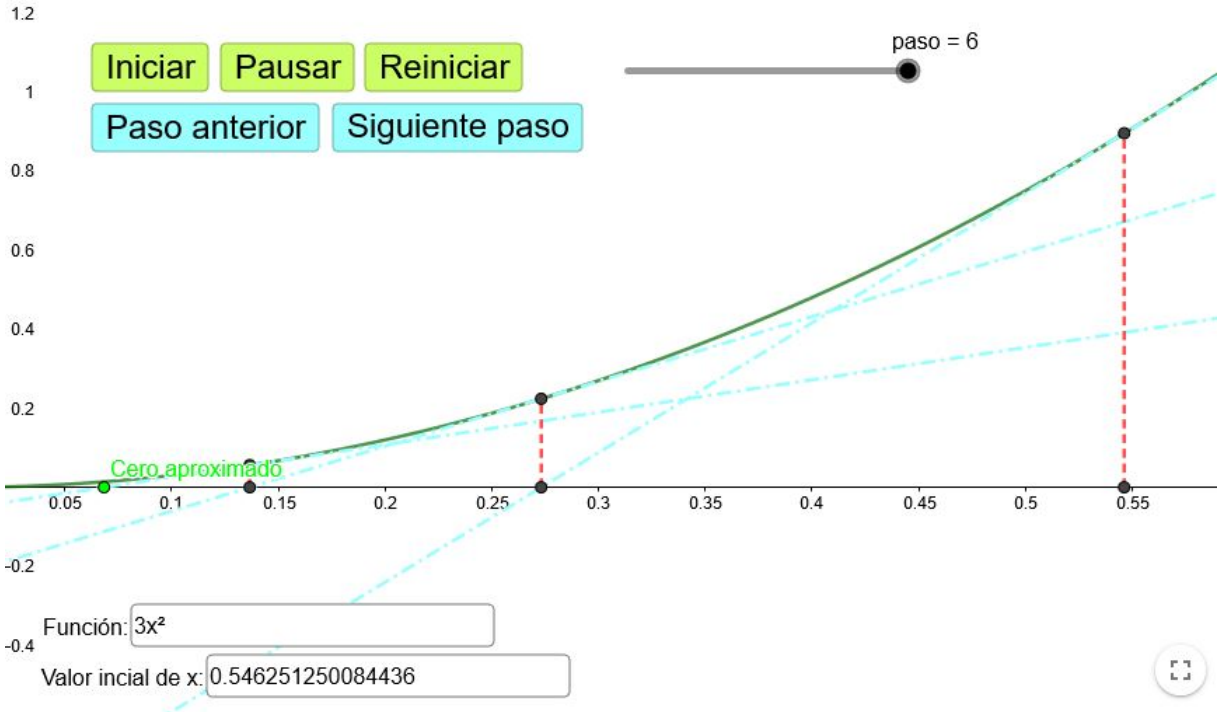


Figura 8: Ejemplo con los 6 pasos efectuados.

Para comenzar a animar el deslizador *paso* se introdujo el botón Ïnciar". Cuando hacemos clic sobre él se ejecuta la instrucción `IniciaAnimación[paso,True]`. Ahora surge la necesidad

de un botón de pausa para que los estudiantes puedan analizar los pasos intermedios, lo que se logra con un botón que al clic ejecute `IniciaAnimación[paso,False]`. Por último, tenemos el botón Reiniciar", que le asigna el valor 0 al deslizador paso con la instrucción `Valor(paso,0)`.

Finalmente, si no se quiere utilizar la animación, quizás el deslizador resulte algo impráctico. Así que también está la opción de avanzar o retroceder de a un paso con otros dos botones. El botón de retroceder ejecuta al clic el comando `Valor(paso,paso-1)`, mientras que el de siguiente ejecuta `Valor(paso,paso+1)` cuando hacemos clic sobre él. Esto permite modificar el valor de *paso* a conveniencia, y por ende lo que se muestra en pantalla.

Interpolación polinomial

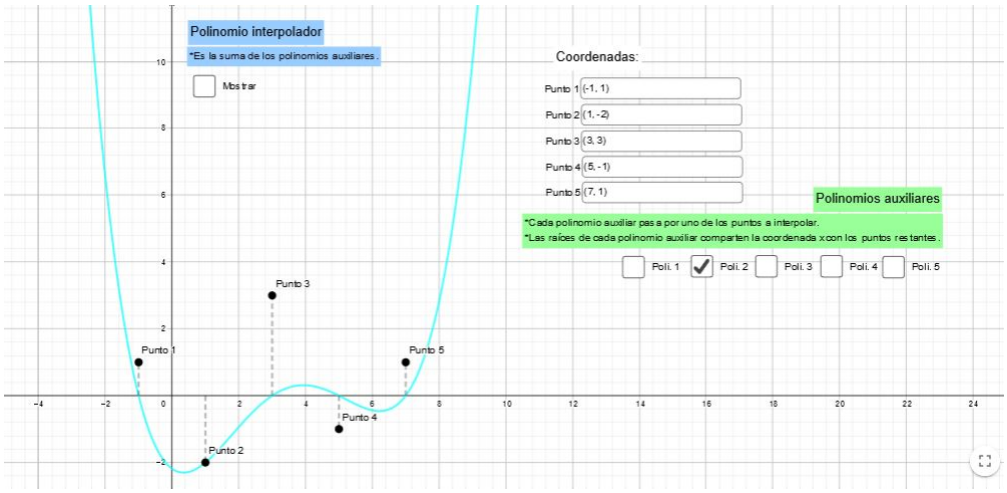


Figura 9: Primer pantallazo del recurso.

El objetivo de este recurso es mostrar cómo se puede construir un polinomio interpolador para el caso de 5 puntos elegidos por el usuario.

Para construirlo, primero armamos 5 puntos A, B, C, D y E con sus respectivos rótulos enumerándolos para distinguirlos. También agregamos casillas de entrada para que el usuario pueda cambiar sus coordenadas.

Luego, definimos los 5 polinomios auxiliares. Recordemos que usando `x(A)`, por ejemplo, obtenemos la coordenada x del punto A. Podemos usar esta herramienta para completar los polinomios auxiliares. Por ejemplo, para el punto A, el polinomio es:

$$\text{Polinomio auxiliar } 1 = \frac{y(A)((x - x(B))(x - x(C))(x - x(D))(x - x(E)))}{((x(A) - x(B))(x(A) - x(C))(x(A) - x(D))(x(A) - x(E)))}$$

y de manera similar definimos los 4 restantes. También armamos el polinomio interpolador que es la suma de los polinomios auxiliares.

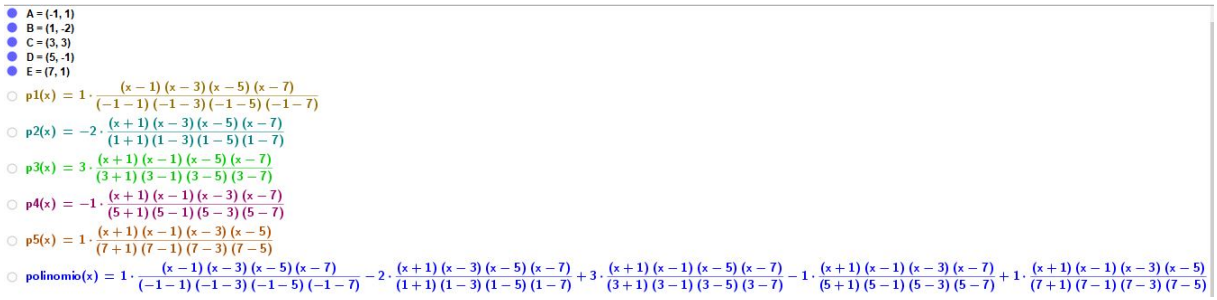


Figura 10: Ejemplo de cómo se verían los polinomios en la vista algebraica para determinados puntos.

Posteriormente, ponemos casillas de verificación para poder mostrarlos u ocultarlos para que el estudiante comprenda mejor el proceso de construcción del polinomio interpolador.

Luego, para que sea evidente que cada polinomio auxiliar tiene raíces con coordenadas x iguales a los puntos por los que no pasa el mencionado polinomio, creamos puntos con coordenadas $(x_{A,B,C,D,E}, 0)$. Estos nos servirán para trazar 5 segmentos punteados que los conecten con A, B, C, D y E mediante la instrucción `Segmento(punto inicial, punto final)`. Una vez formados los segmentos, ocultamos estos puntos para dejar sólo los principales.

Linealización

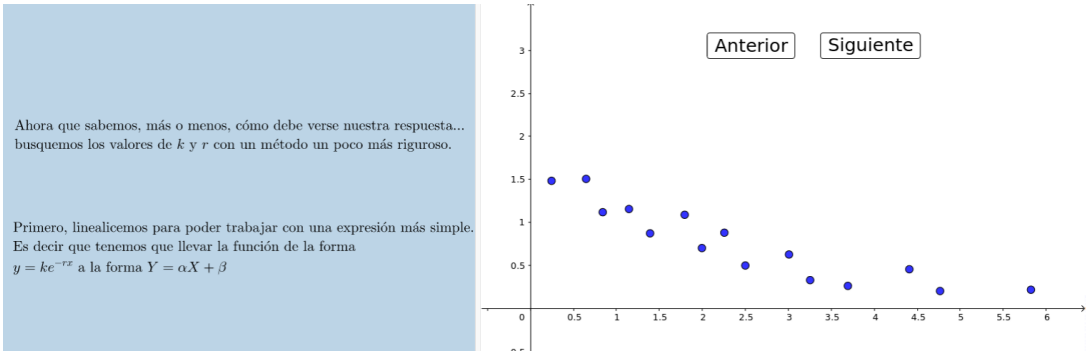


Figura 11: Primer pantallazo del recurso.

En este recurso se utilizan 2 vistas gráficas, una para mostrar texto (explicaciones) y la otra para mostrar los datos y las aproximaciones. Definimos un deslizador `n` que varia de 1 a 11, se incrementa con cada click en *siguiente* y se reduce con cada click en *anterior*. Diferentes objetos (textos, gráficas, etc) se van mostrando u ocltando dependiendo del valor que tome el deslizador.

La parte mas destacable del recurso es la animación de las transformaciones de los datos y la función que los aproxima, donde deseamos que una vez apretado siguiente, pasaran unos segundos para que el usuario pudiera leer el texto y luego comenzara la animación.

Para ello colocamos un condicional en el botón siguiente:

```
Si(n == 6, iniciar la animación de i),
```

`i` es un deslizador continuo que se mueve entre 0 y 1 y al llegar a 1 comenzaría la animación de `t`, otro deslizador continuo que controla la transformación.

La transformación es controlada por `t` de la siguiente forma: dada una función f y una función g, la función (dependiente del parámetro t) $h(x) = tg(x) + (1-t)f(x)$ es igual a f cuando $t = 0$ e igual a g cuando $t = 1$, en $0 < t < 1$ pasa por los estados intermedios.

Algo similar sucede con las listas: dada una lista $a = \{a_n\}$ y una función f, la instrucción: `b = f(a)` define una nueva lista b donde $b_n = f(a_n)$, ésta es la lista transformada por la transformación f. Luego, la instrucción:

```
c = t * b + (1 - t) * a
```

define una lista c con las mismas propiedades que la función h antes mencionada: es igual a la lista a cuando $t = 0$ e igual a b cuando $t = 1$.

La función y los datos son controlados por parámetros (deslizadores) `i` y `t` distintos porque ambas se mueven en diferentes momentos. Un detalle destacable: para asegurar que los deslizadores se detuvieran en 0 o en 1 (teníamos el problema de que se detenían en 0.99) agregamos las instrucciones:

```
Si(t == 0, detener animación de t y Valor(t, 0))
Si(t == 1, detener animación de t y Valor(t, 1))
```

Integrales por rectángulos

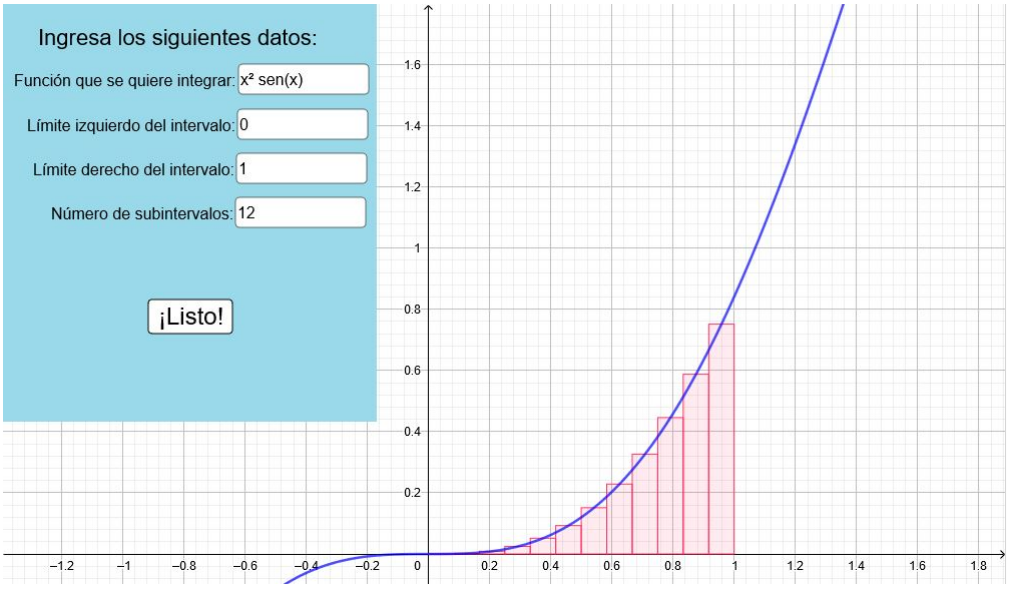


Figura 12: Primer pantallazo del recurso.

El objetivo de este recurso es que el estudiante pueda explorar visualmente las diferencias entre los distintos métodos de aproximación numérica de una integral y pueda ver las ventajas y desventajas de cada uno.

Para construirlo comenzamos definiendo los siguientes objetos:

Casillas de entrada vinculadas a la función y los números:

- `f` Guarda la función ingresada por el usuario.
- `LimiteIzq` Es el número que indica el límite de integración inferior.
- `LimiteDer` Guarda el límite superior.
- `Subintervalos` Almacena la cantidad de subintervalos en que el usuario desea dividir al intervalo principal.

Listas

- `guardamospuntos` es una lista que guardará los máximos de la función en cada subintervalo. Al inicio la definimos como $\{(1,0)\}$.
- `guardamospuntos_{min}` es otra lista que guardará los mínimos de la función en cada subintervalo. Al inicio la definimos como $\{(1,0)\}$.
- `guardamospuntosreset` es una lista de un único elemento que no modificaremos a lo largo de la programación del recurso y está definido como $\{(1,0)\}$. Servirá para limpiar datos anteriores de la lista de máximos.
- `guardamospuntosreset_{min}` es análoga a la lista anterior, pero cumple su función para la lista de mínimos.

Deslizador

- `j` que va de 1 a `Subintervalos`. Nos permitirá desarrollar procesos iterativos cumpliendo la función de contador.

Texto

'Ingresa los siguientes datos:' que irá previo a las casillas de entrada y al botón de inicio.

Antes de comenzar, calculamos `h` haciendo `(LimiteDer - LimiteIzq) / Subintervalos`. Es la longitud de cada subintervalo.

También definimos la lista `intervalos` como `Secuencia(LimiteIzq, LimiteDer, h)`. Es decir que va desde el limite inferior al superior a un paso de h.

Botón "¡Listo!"

Con este botón controlamos la visibilidad de diversos objetos con el comando `Visibilidad(objeto,1,true/false)`. Ocultamos el primer texto junto con las casillas de entrada y mostramos los botones con las 5 opciones y un nuevo texto que nos recuerda cuales fueron nuestras elecciones en las casillas de entrada.

El texto que se muestra ahora es: Quiero integrar `f` desde `LimiteIzq` hasta `LimiteDer` con `Subintervalos` subintervalos utilizando el método de:

Para poder escribirlo incorporamos los objetos señalados en la pestaña 'objeto' del editor de texto. Esto permite que se muestren los valores guardados como números. También recibimos las opciones en los distintos botones de la siguiente manera:

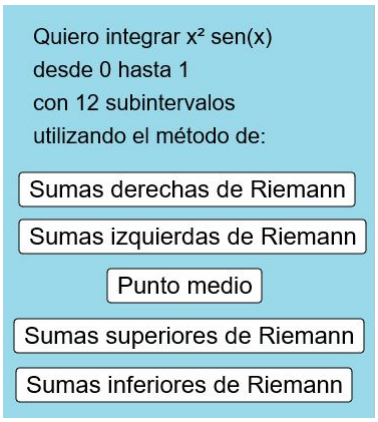


Figura 13: Segunda pantalla de opciones.

Todos los métodos están basados en la misma idea: usar el comando `Histograma` para calcular la aproximación de la integral. Los parámetros que recibe son los límites de los intervalos y luego las alturas de cada rectángulo. En todos los casos, el primer parámetro será la lista `intervalos` que definimos previamente. Ahora veamos cómo manejar las alturas, que controlaremos según el botón clickeado.

Con la acción de uno de los botones, debemos considerar que hay que ocultar los demás ya que son opciones que el usuario descartó. Se debe mostrar el texto del resultado junto con un nuevo botón para regresar. Esto lo veremos posteriormente.

Sumas derechas

Primero armamos una lista de valores de x a la que llamamos `sumasderechas` en la que hay que evaluar la función para poder aproximar cada sub-integral con el comando `Secuencia(LimiteIzq + h, LimiteDer, h)` de modo que posteriormente podamos calcular el valor requerido en el extremo derecho de cada subintervalo. Esta lista se arma siempre, independientemente del botón elegido.

Luego, usamos el área del histograma hecho con el comando `Histograma(intervalos, f(sumasderechas))` como nuestra aproximación.

La acción del botón utiliza el comando `Visibilidad` para mostrar este histograma y ocultar los de los otros métodos.

Finalmente, asignamos el valor del área a una variable llamada `integralcita`. Nos servirá más tarde para calcular el error cometido.

Sumas izquierdas

Similarmente, armamos la lista de valores de x `Secuencia(LimiteIzq,LimiteDer-h,h)` a la que llamamos `sumasizquierdas`.

Usamos el comando `Histograma(intervalos, f(sumasizquierdas))` como nuestra aproximación, lo mostramos ocultando los otros histogramas y guardamos su valor en la variable `integralcita`.

Punto medio

Análogamente a los dos anteriores:

Armamos la lista de valores del eje x haciendo `Secuencia(LimiteIzq+h/2,LimiteDer-h/2,h)` y la llamamos `puntomedio`.

El histograma es `Histograma(intervalos, f(puntomedio))`.

Sumas superiores

Todo se controla con el botón 'Sumas superiores de Riemann'.

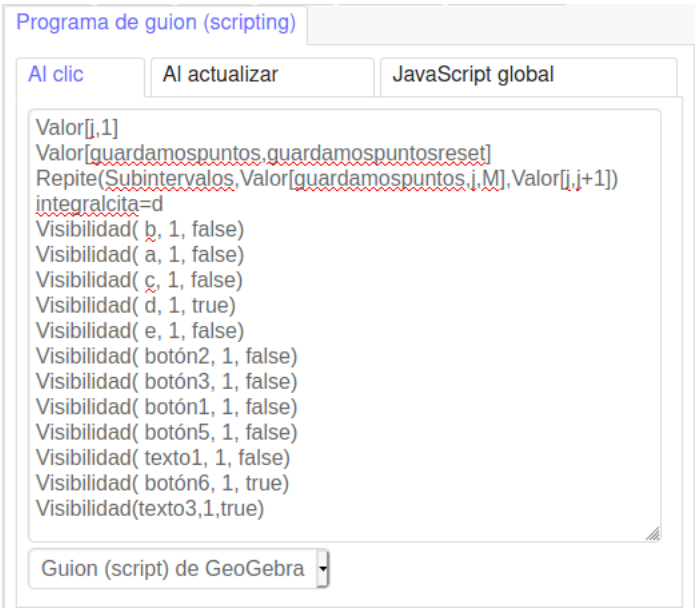


Figura 14: Programación del botón.

Recordemos que j es un deslizador que va desde 1 hasta *Subintervalos*, lo vamos a usar como contador para una tarea que necesitamos que se repita la cantidad de veces que indique *Subintervalos*.

- `Valor(j,1)` reinicia el contador j.
- `Valor(guardamospuntos,guardamospuntosreset)` hace que la lista que usamos para guardar los valores de x para el histograma esté en su forma más simple. Con un solo elemento. Todo esto por si no es la primera vez que el usuario utiliza el método, así que hay que "limpiar"todos los datos previos.
- `Repite(Subintervalos, Valor(guardamospuntos, j, M), Valor(j,j+1))` El comando `Repite` nos permite repetir acciones la cantidad de veces que diga *Subintervalos*. Las acciones a repetir son: Modificar el valor del j-ésimo elemento de la lista *guardamospuntos* por el máximo de la función entre *aa* y *bb*. M está definido como `Máximo(f,aa,bb)`. *aa* está difinido como `Sumasizq(j)` y *bb* como `Sumasderechas(j)`. Aprovechamos estas dos listas que ya teníamos armadas y nos sirven para dar el extremo inicial y final de un mismo subintervalo. Finalmente, actualizamos el contador al próximo número.
- `integralcita=d` le asigna a integralcita el valor d que es del histograma correspondiente a esta integral. (definido aparte como `Histograma(subintervalos,Maximos)`)

Pequeño detalle a tener en cuenta: *guardamospuntos* es una lista de puntos con ambas coordenadas iguales. Quedó definido así porque pensamos que se guardarían los puntos máximos y no el valor máximo. Como para el comando `Histograma` necesitamos extraer un solo número, definimos una nueva lista `Maximos={y(guardamospuntos)}`.

Luego hacemos el histograma con `Histograma(intervalos,Maximos)` y activamos su visibilidad, ocultando a los demás histogramas de los otros métodos.

Sumas inferiores

Con el método de sumas inferiores es exactamente igual que con superiores (salvo que con otras listas, así que por eso definimos $guardamospuntos_{min}$ y $guardamospuntosreset_{min}$). Lo que cambia es que en lugar de buscar el máximo valor entre *aa* y *bb* buscamos el mínimo.

Una vez elegido el botón...

Se muestra el texto 'El valor aproximado es `integralcita` El valor real es `integralposta` así que el error es de `error`.' Lo escribimos seleccionando los objetos correspondientes en la pestaña objetos de modo que se muestren sus valores integrados al texto.

A `integralcita` se le había asignado área del histograma según el botón que seleccionamos.

`integralposta` se calcula con el comando `Integral(f,LimiteIzq,LimiteDer)`.

El botón volver sólo controla visibilidades, no cambia ningún valor. Es el responsable de ocultarse a sí mismo, al botón del método elegido y al texto respuesta; a su vez, muestra la pantalla principal completa con todos sus elementos originales.