Errores de Redondeo y de Truncamiento
Parte II

Profesor: Dr. Alejandro S. M. Santa Cruz
JTP: Dr. Juan Ignacio Manassaldi
Auxiliar: Srta. Amalia Rueda

- En algunos algoritmos, pequeños errores de redondeo pueden conducir a errores de O(1) (100%).
- No obstante, en ciertos casos, esos errores se podrían disminuir cambiando el algoritmo de cálculo.

Ejemplo 1

- Supongamos que queremos evaluar la función $f(x) = e^{-x}$
- Podemos utilizar la aproximación que nos provee la serie truncada de Taylor,

$$f(x) \simeq f(x_0) + hf^{(1)}(x_0) + \frac{h^2}{2!} f^{(2)}(x_0) + \frac{h^3}{3!} f^{(3)}(x_0) + \frac{h^4}{4!} f^{n}(x_0) + \cdots + \; ; \; h = x - x_0$$

- En nuestro caso, desarrollamos la función alrededor del origen, esto es, en $x_0 = 0$, utilizando el desarrollo de Mac Laurin,

$$e^{-x} = \sum_{n=0}^{\infty} \frac{(-x)^n}{n!} = 1 - x + \frac{x^2}{2} - \frac{x^3}{6} + \frac{x^4}{24} - \cdots \; ; \; x \rangle 0$$

- Esta serie tiene un radio de convergencia infinito, esto es, converge para cualquier valor real de x.

Ejemplo 1 (cont.)

- ¿Cuál es el error que cometemos al sumar cada término de la serie?
- Supongamos que cada término en la serie introduce un error absoluto:

$$E = \varepsilon \; x \; magnitud\ de\ cada\ término$$

donde ε representa al épsilon de la computadora.

- Luego, si consideramos que todos los errores se suman (nos ponemos en la peor situación), entonces:

$$E_{total} \simeq \varepsilon + \varepsilon\, x + \varepsilon \frac{x^2}{2} + \varepsilon \frac{x^3}{6} + \varepsilon \frac{x^4}{24} + \cdots \simeq \varepsilon\, e^x$$

Ejemplo 1 (cont.)

- Por consiguiente, resulta:

$$\left(e^{-x}\right)_{numérico} = e^{-x} \pm \varepsilon\, e^{x} = e^{-x}\left(1 \pm \varepsilon\, e^{2x}\right)$$

- Esto es, $\varepsilon\, e^{2x}$, representa al error relativo que se comete al evaluar la función como serie alternante.
- Si evaluamos la serie con Scilab utilizando doble precisión, donde $\varepsilon = 2.220e\text{-}16$ resulta que:
  - Para $x = 1$ el error es pequeño
  - Para $x = 20$ el error relativo es casi del 200%

Ejemplo 1 (cont.)

- Obsérvese que el desarrollo en serie infinita de potencias es exacto y converge (eventualmente) para todo x.
- Pero, debido a los errores de redondeo se obtiene un error grande para x = 20 (~200%)
- La conclusión es: No diseñar algoritmos donde en una serie se restan números grandes para obtener números pequeños.

Ejemplo 1 (cont.)

- ¿Cómo resolvemos esta situación?
- Fácil, evaluamos $f(x) = e^{-x}$ para $x > 0$ como:

$$e^{-x} = \frac{1}{e^x} = \frac{1}{\sum_{n=0}^{\infty} \frac{(x)^n}{n!}}$$

- En este caso, el error relativo del cociente es:

$$error_{rel}\,(1/e^x) = error_{rel}\,(1) + error_{rel}\,(e^x) \sim O(\varepsilon)$$

- Ya que el 1 no introduce error y

$$(e^x)_{num.} = e^x \pm \varepsilon\, e^x = e^x(1 \pm \varepsilon)$$

Overflow y Underflow

- En español, la terminología sería *desbocamiento por exceso* (overflow) o *desbocamiento por defecto* (underflow).
- En algunas oportunidades la elección de un determinado algoritmo puede conducir a problemas de desbocamiento.
- Para entender y solucionar este inconveniente se presentará a continuación un ejemplo que implica la evaluación de una función de probabilidad.

Overflow y Underflow (cont.)

- Supongamos que la probabilidad de ocurrencia de un evento sea p.
- Por ejemplo, si revoleamos una moneda al aire, la probabilidad de obtener cara o cruz es del 50% ($p=0.5$).
- Supongamos ahora que diseñamos un experimento que implica revolear la moneda N veces.
- ¿Cuál es la probabilidad que K veces, con $K \leq N$, obtengamos un resultado favorable (cara o cruz)?
- Este proceso viene descrito por la *función de distribución binomial*,

$$X\left(N,K,p\right)=\binom{N}{K}p^{K}\left(1-p\right)^{\left(N-K\right)}$$

- Donde: $\binom{N}{K}=\frac{N!}{K!\left(N-K\right)!}=\frac{N\left(N-1\right)\left(N-2\right)\cdots\left(N-K+1\right)}{K\left(K-1\right)\left(K-2\right)\cdots 2.1}$

  representa el coeficiente binomial.

Overflow y Underflow (cont.)

- Resulta natural calcular esta probabilidad iterando sobre K y sobre N, así

$$\lambda_1 = \prod_{i=1}^{K} i \quad ; \quad \lambda_2 = \prod_{i=N-K+1}^{N} i$$

- Por lo tanto,

$$\binom{N}{K} = \frac{\lambda_2}{\lambda_1}$$

- Si hacemos esta operación y N es un número grande podríamos tener problemas para evaluar el coeficiente binomial.
- Supongamos ahora que revoleamos 50 veces la moneda ($N=50$), nos preguntamos cuál sería la probabilidad de obtener 25 veces cara o cruz ($K=25$).

Overflow y Underflow (cont.)

- Entonces, para $N=50$, $K=25$ y $p=0.5$, resulta:

$$X(50,25,0.5) = \binom{50}{25}(0.5)^{25}(1-0.5)^{(50-25)} = \binom{50}{25}(0.5)^{25}(0.5)^{25}$$

- Si tratamos de evaluar esta probabilidad calculando λ_1 y λ_2 en simple precisión, obtendremos números realmente grandes,

$$\lambda_2 = \prod_{i=K+1}^{N} i \quad = 1.96x10^{39} \quad Overflow!$$

- Si hacemos esta operación en Scilab o MATLAB, que operan en doble precisión, el resultado sería:

$$X(50,25,0.5) = 0.112$$

- En porcentaje, esa probabilidad sería del 11.2%

Overflow y Underflow (cont.)

- Si N es suficientemente grande, aún trabajando en doble precisión, también podría ocurrir un *overflow* (o eventualmente un *underflow*).
- ¿Como se resuelve este inconveniente o dicho en otros términos como modificaría mi algoritmo para preservarme de estas situaciones?
- En aquellas instancias de cálculo en las que podría ocurrir esta situación deberíamos agregar líneas de código que verifiquen si el número que se genera es demasiado grande (pequeño). Entonces lo dividimos (multiplicamos) por un número grande para mantenerlo dentro del rango de números que la computadora puede almacenar.
- Si conservamos la traza del número de veces que hicimos esto, entonces podemos recuperar el resultado final.

Problemas Mal Condicionados

- En muchas ocasiones los problemas matemáticos que se generan están mal condicionados (o acondicionados).
- En estos casos los errores de redondeo tienden a crecer de manera exponencial y no hay algoritmo numérico que los resuelvan.
- Supongamos que queremos determinar las raíces del siguiente polinomio:

$$\left(x-\frac{2}{3}\right)^4=0$$

- Resulta obvio que la raíz (valor que hace cero la función) de este polinomio es $2/3=0.66666\ldots$). Además es raíz múltiple de multiplicidad 4.

Problemas Mal Condicionados (cont.)

- Supongamos ahora que atacamos el problema desarrollando el polinomio, esto es, expresándolo en términos de las potencias de x, así,

$$x^4 - \frac{8}{3}x^3 + \frac{24}{9}x^2 - \frac{32}{27}x + \frac{16}{81} = 0$$

y almacenamos los coeficientes en simple precisión.

- Si resolvemos mediante algún método de determinación de raíces de polinomios encontramos lo siguiente:

$$x_1 = 0.6861$$

$$x_2 = 0.6663 + 0.0191i$$

$$x_3 = 0.6663 - 0.0191i$$

$$x_4 = 0.6480$$

$$con\ i = \sqrt{-1}\ :\ unidad\ imaginaria$$

Problemas Mal Condicionados (cont.)

- En este caso, el error de las soluciones reales es del orden del 2.8%
- El algoritmo me suministró 4 soluciones, 2 raíces reales y 2 complejas conjugadas.
- ¿Porqué? El problema está mal condicionado.
- La computadora resolvió un problema muy parecido al problema original pero que tiene soluciones muy diferentes.
- ¿Que sucede si reemplazamos la solución x_1 en la estructura original del problema?

$$x_1 = 0.6861 \Rightarrow \left(x_1 - \frac{2}{3} \right)^4 = 1.4x10^{-7}$$

lo cual nos indica que el error está dentro del error de redondeo del cero.

Problemas Mal Condicionados (cont.)

- En un problema mal condicionado un pequeño cambio en los valores de los coeficientes trae aparejado un cambio sustancial en la solución.
- La dificultad yace en el problema, no en el algoritmo que se utiliza.
- Consecuentemente, se debe tratar en lo posible de evitar este tipo de problemas.