Simulando leis da física com Go

“Da inércia ao build”

Hotsite de apoio: https://rpagliuca.github.io/da-inercia-ao-build/

Rafael J. Pagliuca dos Santos
Engenheiro Físico e Mestre em Ciências da Computação

YES.Technology: https://yes.technology
LinkedIn: https://www.linkedin.com/in/rafael-j-pagliuca-dos-santos-7a08b9a1/

Tópicos

- Objetivo desta talk (5 minutos)
  - Sobre o que NÃO vou falar
  - Por que Golang?
- Por que simular? (5 minutos)
  - Solução analítica vs. cálculo numérico
- Alguns modelos físicos (10 minutos)
- Engine de física rudimentar em Go (10 minutos)
- Packages e bibliotecas em Go (5 minutos)

Objetivo desta talk

Objetivos principais

1. Tributo à ciência
2. Compartilhar uma visão de alto nível sobre simulação numérica
    - Para desenvolvedores que não tiveram a oportunidade de cursar disciplinas desse tipo na faculdade
3. Mostrar a cientistas, estudantes e outros interessados que Go é uma linguagem de propósito geral
    - Com um bom ecossistema
    - Adequada para simulações e cálculo numérico

Sobre o que vou falar

- Métodos numéricos
- Modelos físicos básicos
- Programação em Go
- Recomendação de bibliotecas em Go

Sobre o que NÃO vou falar

- Game studies
- Game design
- Modelagem e arte 2D/3D
- Efeitos visuais e animação

Por que Go para simulações?

- Facilidade de concorrência
- Segurança de memória
- Simplicidade
- Performance
- Tipagem estática
- Ecossistema rico
  - Documentação
  - Testes automatizados
  - Gerenciador de pacotes
  - CGO

Por que não Go?

- Garbage collection

Por que simular?

Como a Física avança?

1. Observa-se a natureza
2. Modelos físicos são sugeridos para descrever o fenômeno observado
    - Podem ter um domínio de validade
    - Normalmente são descritos por equações diferenciais
3. Equações diferenciais podem ter soluções analíticas (raramente) ou resultados numéricos aceitáveis (com maior frequência)

Como a Física avança? (continuação)

4. Soluções analíticas são confrontadas com resultados experimentais (novas observações da natureza), e os modelos são descartados, aprimorados ou têm o seu domínio de validade reduzido
    - Soluções numéricas podem estar erradas, e o modelo correto
    - Simulações físicas normalmente usam soluções numéricas
5. De volta ao passo 1

Alguns modelos físicos

Alguns modelos físicos

Gravidade ao nível do mar

Leis que descrevem movimento e gravidade

- Segunda lei de Newton

$$F = ma$$

- Lei da gravitação universal de Newton

$$F = G\frac{Mm}{r^2}$$

Constantes para gravidade na Terra ao nível do mar

- Juntando as duas leis:

$$ma = G\frac{Mm}{r^2} \qquad a = G\frac{M}{r^2}$$

$$G = 6.67430 \cdot 10^{-11}$$
$$M = 5.972 \cdot 10^{24}$$
$$r = 6371 \cdot 10^{3}$$

$$a = 9.819973426$$

PS: Unidades do S.I. omitidas por simplicidade

PS2: Raio **r** varia pois a Terra não é uma esfera perfeita

Isso é uma equação diferencial?

- Equação diferencial

$$a = 9.819973426$$

- Solução analítica

$$S = S_0 + V_0 t + \frac{at^2}{2}$$

$$\ddot{x} = a$$

$$\dot{x} = \int_0^t \ddot{x} + V_0$$

$$x = \int_0^t \dot{x} + S_0$$

Código de simulação usando solução analítica

- The Go Playground:
  - https://play.golang.org/p/-bvvrsAm3j6
- **Deu certo! Agora estou pronto para simular qualquer coisa?!**
  - Quase! ;-)
  - Só existem soluções analíticas (fórmulas prontas) para poucos fenômenos físicos bem simples, e normalmente envolvendo poucos objetos (1 ou 2, talvez alguns mais) e condições iniciais bem especiais.
  - Até para alguns casos simples a nossa solução anterior não funciona: quanto tempo demoraria para um objeto a 500km de altura (altura dos satélites Starlink) cair até o nível do mar?
    - $g = 8{,}44\ m/s^2$

Simulação numérica

- Força pode ser convertida em aceleração e vice-e-versa pela segunda lei de Newton

$$F = ma$$

- Aceleração (**a**) pode ser convertida em posição (**x**) resolvendo a equação diferencial

$$\ddot{x} = a$$

Resolvendo a equação diferencial numericamente

- Modelamos o tempo de forma discreta em vez de contínua:

$$t_{n+1} = t_n + \Delta t$$

- Assumimos que o mundo fica parado entre dois instantes discretizados
- Em cada instante de tempo, apenas o objeto se move, e todo o resto do mundo fica constante (incluindo outros objetos)

Código de simulação usando solução numérica

- Forward Euler method
  - https://play.golang.org/p/HW3zZQdsLL8
- Já conseguimos observar erro de precisão...
  - O erro diminui linearmente ao diminuirmos o intervalo de tempo em que o mundo fica parado

Usando Runge-Kutta de ordem 4

- RK4
  - https://play.golang.org/p/Os_7O00sXdn
- Melhor!
  - O erro é proporcional a $\mathcal{O}(\Delta t^4)$

Alguns modelos físicos

Gravidade em grandes altitudes

Gravidade variando com a altura

- Usamos as mesmas leis físicas anteriores
  - Segunda lei de Newton
  - Lei da gravitação universal de Newton
- Novidade: agora recalculamos **g** para cada posição **S** (altura)
- Código:
  - https://play.golang.org/p/g1vpeFFNMZF

E agora com resistência do ar!

- Utilizamos a equação do arrasto:

$$F_D = \frac{1}{2}\rho u^2 C_D A$$

- Assumindo C_D, A e m constantes:

$$a = C\rho v^2$$

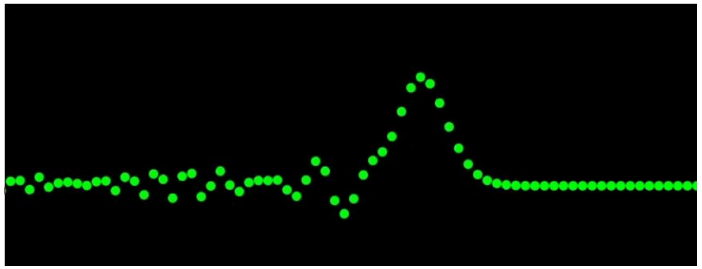
- **rho** é a densidade do ar; **C** deve ser estimado experimentalmente
  - https://www.scielo.br/pdf/rbef/v37n2/0102-4744-rbef-37-02-2306.pdf
- Simulação:
  - https://play.golang.org/p/GKndxvtyQUR

Alguns modelos físicos

Propagação de ondas em 1 dimensão

Equação de onda em 1 dimensão

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}$$



Interpretação: a “aceleração” da intensidade de em um ponto varia ao longo do tempo proporcionalmente à diferença de intensidade em relação aos pontos vizinhos.

Solução analítica

- Existe para condições iniciais simples
- Formada da combinação de senos e cossenos

Solução numérica

- Obtida de maneira análoga ao caso da gravidade
- Diferença: agora nossa função **u** (intensidade) depende de 2 variáveis (**x** e **t**)
  - Antes a nossa função **S** (posição/altura) dependia apenas de 1 variável (**t**)
- Novamente conseguimos resolver discretizando o tempo (e agora o espaço, variável **x**!)
  - Chamamos de “Método das diferenças finitas”
- Simulação:
  - https://play.golang.org/p/YoXuXVIBjEB

Alguns modelos físicos

Equação de difusão do calor

Muito parecida com a equação de ondas

$$\frac{\partial u}{\partial t} = \eta \frac{\partial^2 u}{\partial x^2}$$

Interpretação: a variação no tempo é muito mais lenta do que a equação de onda, já que agora temos a primeira derivada no tempo.

Também é solucionada numericamente pelo método das diferenças finitas.

Engine de física rudimentar em Go

Movimento de objetos

Princípios

- Calcula-se a aceleração individual da interação entre um objeto e todos os outros
  - **Goroutines! Yay!**
- A aceleração total que atua em um objeto é a soma de todas as acelerações individuais
- Até agora trabalhamos em 1 dimensão
  - Migrar para 2 e 3 dimensões não é muito diferente: utilizamos geometria analítica para fazer a decomposição da aceleração nas diferentes coordenadas.

$$F = ma \qquad \vec{F} = m\vec{a} \qquad \begin{aligned} a_x &= \vec{a} \cdot \vec{x} \\ a_y &= \vec{a} \cdot \vec{y} \end{aligned}$$

Geometria analítica

- Produto escalar entre 2 vetores
  - Possibilita calcular componentes de aceleração
- Vetor normal conectando um ponto e reta
  - Permite calcular direção de força, gravidade, etc
- Vetor normal em relação a uma reta
- Intersecção entre 2 segmentos de reta (colisão)
- Normalização de vetores

Código de exemplo

- Gravidade a nível do mar
- Multi-objetos
  - 1 objeto se movendo
  - Múltiplos corpos estáticos (paredes)
- Colisão e força normal entre paredes e corpo que se move
- Código-fonte:
  - https://github.com/rpagliuca/code-samples-da-inercia-ao-build/tree/main/04-engine

Bibliotecas interessantes e referências

Bibliotecas que indico

- Bindings em Go para OpenGL:
  - https://github.com/go-gl/gl
- Biblioteca matemática para operação com vetores e matrizes:
  - https://pkg.go.dev/github.com/go-gl/mathgl
- Renderização vetorial em 2D:
  - https://github.com/fogleman/gg
- Game engine 2D:
  - https://ebiten.org/

Alguns projetos meus

- Helpers para criar uma aplicação mínima com OpenGL em Go:
  - https://github.com/rpagliuca/go-gl-helpers
- Renderização em 2D e 3D de equações de movimento e ondas:
  - https://github.com/rpagliuca/go-physics

https://yes.technology