多功能 Hamilton 力学模拟器与其应用案例

詹有丘

2020 年 9 月 11 日

摘要

我们开发了一个方便的可以模拟 Hamilton 力学系统的在线软件. 只要用户把系统的 Hamiltonian 和初始条件告诉模拟器, 就能看到这一系统的运动. 模拟器还能用 FFT 算法得到运动的频域, 来分析振子的振动模式. 模拟器体积小而速度快, 而且对它进行操作和自定义方便且容易. 用户界面很简单 (图形界面用于简单的基础操作, 控制台界面用于其他操作). 模拟器还能输出被模拟的系统的数据来创建数据集, 用于其他潜在的用途. 有很多能用它完成的应用.

***Keywords*—** hamiltonian, 模拟, 力学, 可视化

目录

1 引入

理论力学是很难学习的科目. 人们经常发现自己很难想象力学系统是什么样的, 从而可能想要看看运动的图像. 力学系统被定义之后, 能够输出系统的运动的软件被称为力学模拟器.

互联网上已经有了很多可以使用的力学模拟器, 但它们大多有以下的缺点之一:

1. 太庞大. 有些模拟器非常强大, 但代价就是它巨大的体积. 这使得它们不便携.
2. 不够方便. 大多模拟器要求用户将程序文件下载到磁盘. 这是不方便的, 因为当你使用另一台设备时你需要重新安装软件.
3. 不够可自定义. 有些模拟器关注于真实生活中的寻常模型. 对于刚体接触的问题, 它们可能很有用. 但是它们通常不带有诸如模拟任意有心力场中的运动或解决狭义相对论问题的功能.
4. 无法输出数据. 有些模拟器专注于当系统变化时将其呈现出来, 但它们缺少方便的接口来输出系统运动的数据.
5. 太难操作. 有些强大的模拟器有非常复杂的用户界面, 这要求用户花费数小时的学习来学会操作模拟器和获取结果. 这对于新用户并不友好.

所以, 我们想要创造一个能解决上面的问题的模拟器. 为了让它方便使用, 它应当被挂载在一个网页上, 这样所有有浏览器的人都能在能访问互联网时使用它.

有了这样一个模拟器, 人们可以更方便地学习理论力学. 人们可以对特定的力学系统有感性的认知. 模拟器还能被用于在物理教育或讲座上呈现动态演示.

2 符号列表

注意, 如果一个符号具有定义域 $A \to B$, 这意味着它是一个从集合 A 到集合 B 的函数, 那么它有时可以表达该函数的值 (在集合 B 中). 换句话说, 如果 $f : A \to B$ 是关于 x 的函数, 那么 f 可以是 $f(x)$ 的简记.

我们总是假设我们遇到的函数具有足够好的性质, 如果我们需要使用这一性质的话.

符号列表由表 1 给出. 数学运算列表由表 2 给出.

所有的量都在计算机程序中以浮点数实现, 这些量并不需要使用和现实中常用的单位相同的单位, 而是使用其他更加方便的单位 (如像素或帧), 所以单位并没有在表中被提到.

有一些模型有特定的符号, 在节 6 中被提到. 它们并没有被包含在符号列表中, 但它们的特别含义在那节中被解释了.

表 1: 符号列表

符号	定义域	意义	值
t	$\mathbb{R}$	时间	
Δt	$\mathbb{R}^+$	ODE 求解器的步长	5×10^{-4}
ι	$\{2\zeta \mid \zeta \in \mathbb{Z}^+\}$	ODE 求解器中向量的维数	
$\mathbf{x}$	$\mathbb{R} \to \mathbb{R}^\iota$	系统的状态, 关于 t	$(\mathbf{q}, \mathbf{p})$
$\mathbf{q}$	$\mathbb{R} \to \mathbb{R}^{\iota/2}$	广义坐标, 关于 t	
$\mathbf{p}$	$\mathbb{R} \to \mathbb{R}^{\iota/2}$	广义动量, 关于 t	
$\mathcal{H}$	$\mathbb{R} \times \mathbb{R}^\iota \to \mathbb{R}$	Hamiltonian, 关于 $(t, \mathbf{x})$	
$\boldsymbol{\omega}$	$\mathbb{R}^{\iota \times \iota}$	用于求辛梯度的矩阵	$\begin{bmatrix} \mathbf{O} & \mathbf{I}_{\iota/2} \\ -\mathbf{I}_{\iota/2} & \mathbf{O} \end{bmatrix}$
ξ	$\mathbb{R}$	画布上的横坐标, 以像素为单位	
η	$\mathbb{R}$	画布上的纵坐标, 以像素为单位	
m_t	$\mathbb{R} \to \mathbb{R}$	从真实的 t 到画布上的 ξ 坐标的映射	
m_y	$\mathbb{R} \to \mathbb{R}$	从真实的 $\mathbf{x}$ 分量到画布上的 η 坐标的映射	
w	$\mathbb{Z}^+$	画面的宽度	1024
h	$\mathbb{Z}^+$	画面的高度	768
y	$\mathbb{R}$	$\mathbf{x}$ 被绘制的分量	
N	$\mathbb{Z}^+$	用于计算 DFT 的样本数	1×10^5
W	$[0, 1) \to \mathbb{R}$	窗函数	见式6

表 2: 运算列表

符号	名称	定义
$\dot{f}$	f 关于 t 的全导数[1]	$\frac{\mathrm{d}f}{\mathrm{d}t}$
Δf	f 的全变化[2], 当 t 变为 $t + \Delta t$	$f(t + \Delta t) - f(t)$
$\sum_j^n r_j$	n 个关于下标 j[3]的数之和	$\sum_{j=0}^{n-1} r_j$
$\zeta \,\%\, \chi$[4]	ζ 除以 χ 的余数	$\zeta - \chi \left\lfloor \frac{\zeta}{\chi} \right\rfloor$

3 物理理论

3.1 动力学预测

一个动力系统的运动可以用形式为

$$\dot{\mathbf{x}} = \mathbf{f}(t, \mathbf{x}) \tag{1}$$

的常微分方程 (ODE) 预测, 其中 $\mathbf{f} : \mathbb{R} \times \mathbb{R}^\iota \mapsto \mathbb{R}^\iota$ 是一个与系统本身有关的特定函数, ι 是某个正整数, 它不必须是系统的自由度 (DOF) (在我们的情形中, 它实际上是 DOF 的两倍).

3.2 ODE 求解器

该 ODE (式 1) 能用 Runge–Kutta 方法[5]

$$\Delta\mathbf{x} \approx \Delta t \sum_j^s b_j \mathbf{K}_j, \tag{2}$$

被数值求解, 其中 $\mathbf{K}_j$ 被递推地定义为 [9, p. 907]

$$\mathbf{K}_j := f\left(t + \Delta t \sum_k^j a_{j,k}, \mathbf{x} + \Delta t \sum_k^j a_{j,k}\mathbf{K}_k\right). \tag{3}$$

Δt 越小, 求解器就越精确而低效.

阶数 s 和系数 b_j 与 $a_{j,k}$ 是不同的 Runge–Kutta 方法特有的. 这里, 3/8 准则 [4, p. 138] 被采用. 它的系数在表 3 中被展示.

表 3: Runge–Kutta 方法 3/8 准则的系数

	$a_{j,k}$				b_j
j \ k	0	1	2	3	
0					$1/6$
1	$1/3$				$1/3$
2	$-1/3$	1			$1/3$
3	1	-1	1		$1/8$

[1]全导数意味着: 若 f 是关于 g 的函数, g 是关于 t 的函数, 那么 $\dot{f}$ 是 $\frac{\mathrm{d}}{\mathrm{d}t} f(g(t))$.

[2]全变化类似于全导数. 见脚注 1.

[3]按计算机中的习惯, 下标从 0 开始, 而不是 1. 这一习惯在本文中被遵守.

[4]这一记法来自计算机的习惯.

[5]该算法能用 Ruby 编程语言更简洁地描述:

```ruby
dx = b.zip(a).each_with_object([]).sum do |(bj, aj), ary|
  bj * ary.push(f.(t+aj.sum*dt, x+aj.zip(ary).sum{_1*_2}*dt)).last
end * dt
```

ODE 求解器应当保存 t 和 $\mathbf{x}$, 而且每当它前进一步, 记录 $(t, \mathbf{x})$, 并令 $\mathbf{x} \leftarrow \mathbf{x} + \Delta\mathbf{x}$, 且 $t \leftarrow t + \Delta t$.

ODE 求解器能给出任意 t 处 $\mathbf{x}$ 的数值, 只要 $\mathbf{f}$ 的数值形式和初值 $\mathbf{x}(0)$ 被给定.

3.3 构建 ODE

仅仅 ODE 求解器对模拟动力学系统没有帮助. ODE 本身是必须的. 根据物理理论, 有很多构建动力学系统的 ODE 的方法. 这里, Hamiltonion 方法被采用.

Hamilton 力学认为, 对于某个动力学系统, 存在函数 $\mathcal{H} : \mathbb{R}^\iota \times \mathbb{R} \to \mathbb{R} : (t, \mathbf{x}) \mapsto \mathcal{H}(t, \mathbf{x})$ 使得该系统的运动满足被称为**正则方程**[6]的方程

$$\dot{\mathbf{x}} = \boldsymbol{\omega}\nabla_{\mathbf{x}}\mathcal{H}, \tag{4}$$

其中 $\boldsymbol{\omega}\nabla_{\mathbf{x}}$ 是关于 $\mathbf{x}$ 的**辛梯度**.

在 Hamiltonian 力学的情形下, 矢量 $\mathbf{x} \in \mathbb{R}^\iota$ 可以被拆分为两个矢量 $\mathbf{q} \in \mathbb{R}^{\iota/2}$ 和 $\mathbf{p} \in \mathbb{R}^{\iota/2}$, 分别被称为系统的**广义坐标**和**广义动量**, 所以 $\mathbf{x}$ 的分量可以被叫做 `q0`, `q1`, `p0`, `p1`, 等等.

因为梯度 $\nabla_{\mathbf{x}}\mathcal{H}$ 可以容易地被数值计算, 根据

$$\mathbf{f}(t, \mathbf{x}) := \boldsymbol{\omega}\nabla_{\mathbf{x}}\mathcal{H}, \tag{5}$$

我们可以给出式 1 中 $\mathbf{f}$ 的数值形式, 从而根据节 3.2 中描述的方法数值求解式 1.

3.4 分析运动的频域

当我们研究动力学系统的周期运动, 研究它的频域通常是有趣的. 所以, 我们想要让模拟器带有展示运动的 Fourier 变换 (FT) (在一段时间 $[0, N\Delta t)$ 上的) 的能力. 因为我们数值地完成这件事, 而且 t 实际上是离散的, 所以我们实际上计算的是离散 Fourier 变换 (DFT).

在节 4.2 中提到的 FFT 库提供了计算 DFT 的方法. 虽然我们可以直接挑一个足够长的区间来计算它的 DFT, 这一操作会导致一些频域中的损失 [6]. 在计算 DFT 之前, 我们应当在该区间上给运动应用一个窗函数.

有各种各样的窗函数可以选择, 每一种都有它特有的应用场景. 这里, Hamming 窗 [6]

$$W(\zeta) := \frac{25}{46} - \frac{21}{46}\cos(2\pi\zeta) \tag{6}$$

被应用, 因为它适合大多数情况.

4 被模拟器使用的库

我们的模拟器依赖于一些第三方库.

图形库被用于在屏幕上显示图像.

FFT 库被用于计算系统运动的 DFT.

ODE 构建器和 ODE 求解器是我们写的, 不依赖第三方库. 它们的理论基础在节 3 中被解释.

[6] 正则方程通常在其他书 [5][2, p. 65][8, p. 132] 中被记作

$$\dot{\mathbf{q}} = \frac{\partial\mathcal{H}}{\partial\mathbf{p}}, \quad \dot{\mathbf{p}} = -\frac{\partial\mathcal{H}}{\partial\mathbf{q}}.$$

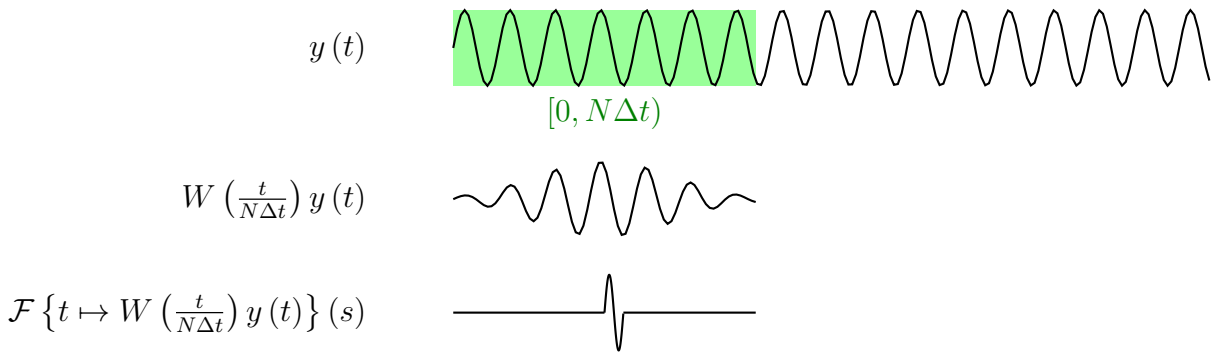


图 1: 获取频域的过程

4.1 图形库

我们使用 rpg_core.js 来绘制和显示图像. 这是一个基于PixiJS的网页游戏引擎. 虽然 rpg_core.js 附属于非免费的软件RPG Maker MV, 但它在GitHub上被开源.

在 rpg_core.js 中, `Bitmap` 对象被用于存储一幅图片的信息, 而 `Sprite` 对象被用于呈现被 `Bitmap` 对象描述的图片. 坐标等信息也在 `Sprite` 对象中 [1]. 图 2 展示了 rpg_core.js 如何显示图片.

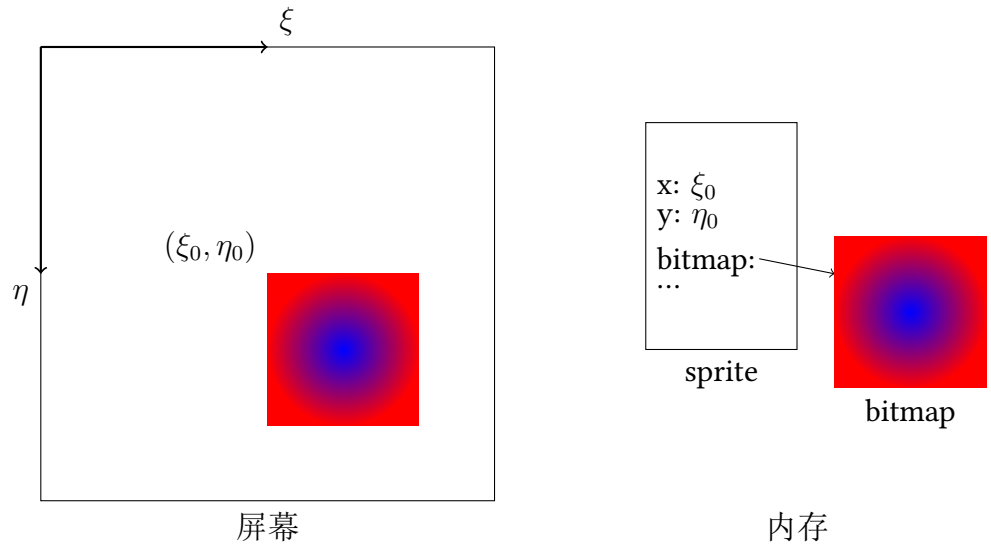


图 2: rpg_core.js 如何显示图片

`Sprite` 对象的 `x` 和 `y` 属性可以被修改, 用于移动图片.

rpg_core.js 还提供了在 `Bitmap` 对象上用某个颜色填充矩形区域的方法. 这使我们能设置 `Bitmap` 对象上像素的颜色, 从而绘制图像.

4.2 FFT 库

FFT 库实现快速 Fourier 变换 (FFT) 算法. 我们使用的 FFT 库是用 C 编写的 FFTW. 因为我们要将它用在网页上, 我们使用了 emscripten 来引入它.

5 绘制图像

根据节 3 中描述的理论, 可以设计程序来给出任意 t 处的广义坐标 $\mathbf{q}$ 和广义动量 $\mathbf{p}$, 根据输入的 Hamiltonian $\mathcal{H}$ 与初值 $\mathbf{q}(0)$ 和 $\mathbf{p}(0)$.

然而, 人很难找出运动的规律仅仅通过观察很多 $(t, \mathbf{x})$ 对. 为了让找出运动规律更简单, 模拟器应当能够根据 $(t, \mathbf{x})$ 对绘制图像.

更具体地, 对于 $\mathbf{x}$ 的每个分量 y, 在 ξ-η 平面 (画布) 上, 描点 $(m_t(t), m_y(y))$. 引入 m_t 和 m_y 是因为画布上的坐标以像素为单位, 这是一个很小的单位. 另一个引入 m_t 和 m_y 的目的是让用户能够使用诸如对数尺度的非线性尺度.

5.1 卷动图像

因为人们通常想要在一长段时间内模拟一个系统, 这将让图像非常宽, 所以画布必须比屏幕宽得多. 于是在模拟器模拟系统的时候我们必须让画布卷动.

虽然如今计算机可以在屏幕上快速地绘制图像, 能够在 1/60 秒内重新绘制屏幕, 但是对 `Bitmap` 对象的读写操作是耗时的. 所以, 我们想要实现类似于 Carmack 卷动算法的卷动算法. 使用这一方法, 每当 ODE 求解器前进一步时, 计算机只需要在 `Bitmap` 上改变一个像素, 而不是几万个像素

该算法需要两个 `Sprite` 对象, 分别叫做精灵 1 和精灵 2, 他们需要分别显示一幅宽 w 高 h 的 `Bitmap` 对象 (所以总共有两个 `Bitmap` 对象), 其中 w 和 h 也是显示图像的屏幕的宽度和高度.

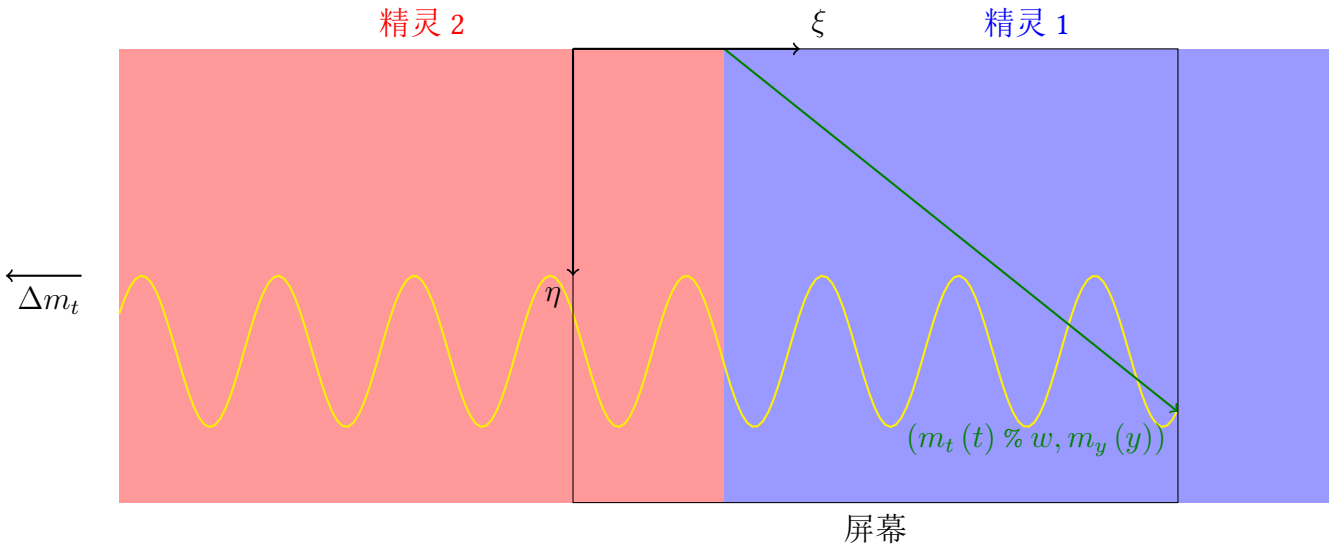
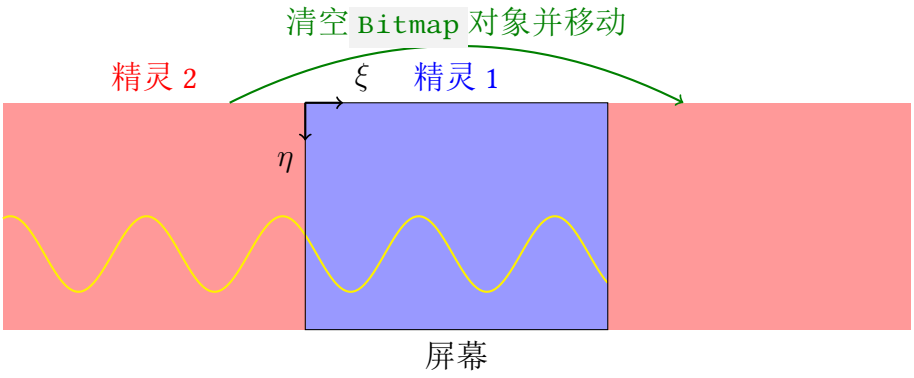


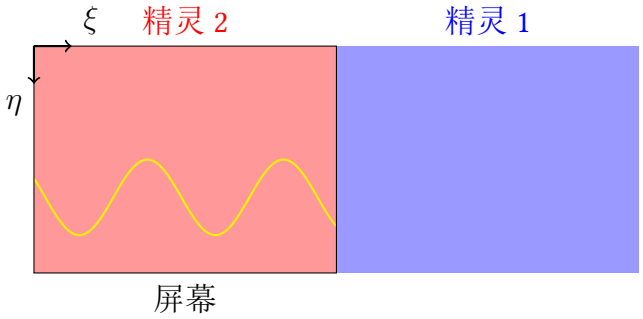
图 3: 当 ODE 求解器推进时, `Sprite` 对象如何移动, 以及 `Bitmap` 对象如何被绘制

当 ODE 求解器推进时, 精灵 1 和精灵 2 向左移动 Δm_t. 现在, 精灵 1 的坐标是 $(w - (m_t(t) \,\%\, w), 0)$, 精灵 2 的坐标是 $(-(m_t(t) \,\%\, w), 0)$. 在精灵 1 的 `Bitmap` 对象上填充位于 $(m_t(t) \,\%\, w, m_y(y))$ 的像素. 这一过程被图 3 展示.

当两个精灵向左移动得足够多了之后, 精灵 1 接触到屏幕的左端. 此时, 精灵 2 突然移动到精灵 1 的右侧, 清空其 `Bitmap` 对象上的所有像素, 然后和精灵 1 交换名字. 这个过程被图 4 展示.



(a) 精灵 2 突然清空其 `Bitmap` 对象并移动



(b) 精灵 1 和精灵 2 交换名字

图 4: 精灵如何突然移动和交换名字

5.2 展示运动的频域

在节 3.4 中提到, 我们需要分析运动的频域. 因为振动的频域通常是离散而稀疏的, 而且高频区域通常是几乎为零的, 最好将低频区域沿着屏幕的宽度延展, 然后将相邻的频域图像上的点用线段连接起来.

因为 rpg_core.js 不带有在 `Bitmap` 对象上画直线的方法, 我们需要实现画直线的算法. 这里, Bresenham 的直线算法被采用.

6 操作指南

当你打开网页, 它会开始模拟默认的模型, 这是带有交变外力 [8, p. 61] 的一维参变振动 [8, p. 82]

$$\mathcal{H}(t,q,p)=\frac{p^2}{2}+\omega^2\left(1+u\cos(\gamma t)\right)\frac{q^2}{2}-fq\cos(\kappa t+\beta), \tag{7}$$

其中

$$(u,\gamma,\beta,\kappa,\omega,f)=(0.3,21.7,0.2,8,10,20),$$

初条件为

$$(q,p)=(2,0).$$

频域计算功能被启用. 一段时间后, 当积累了足够多的样本, 模拟器会在屏幕左上角创建按钮, 点击按钮可以让模拟器显示将被 FFT 的函数时域以及计算出的频域结果.

在频域的界面, 和按钮颜色相同的线表征 FFT 结果的实部, 灰色的线表征 FFT 结果的虚部.

按 [ ] 以暂停或继续模拟.

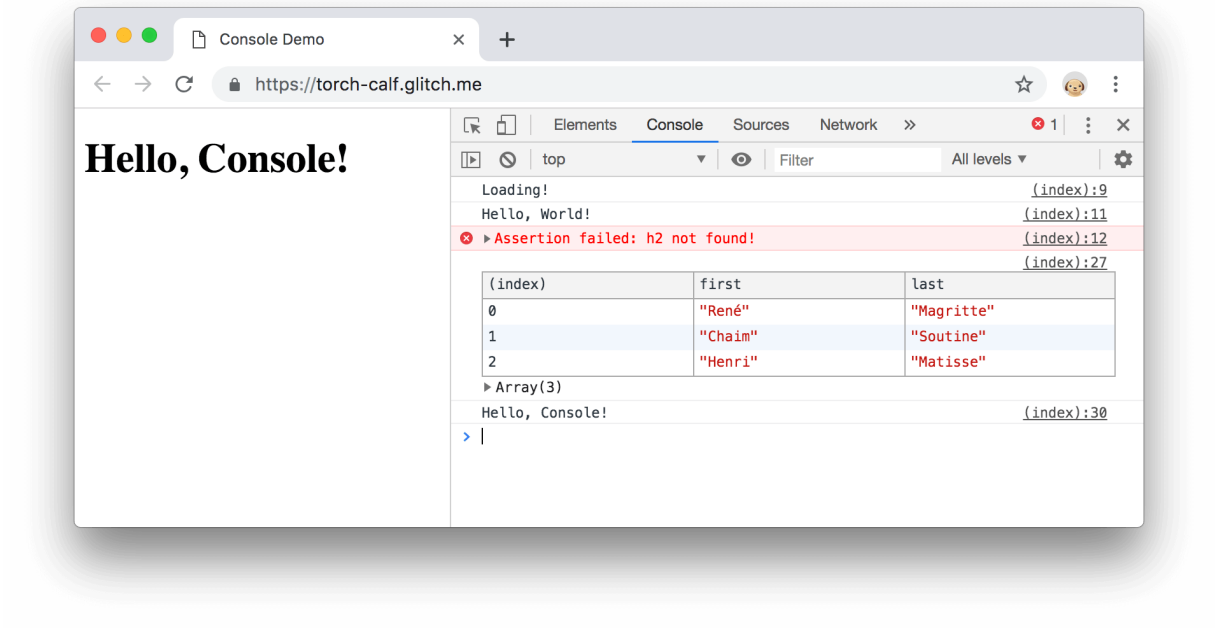


图 5: Chrome 浏览器的控制台界面 (英文) [3]

图 6 展示了模拟器的图形界面. 从模拟结果看, 我们确实能学到一些运动规律. 频域图像很清晰.

如节 1 中所介绍, 模拟器可以被高度自定义. 你能通过在控制台[7]写代码来自定义模拟器.

6.1 下载运动数据

模拟器默认不会记录 ODE 求解器的历史. 为了让模拟器记录历史, 运行下面的代码.

```
rungeKutta.recordHistory = true;
restart();
```

等待 ODE 求解器积累足够的数据, 然后运行下面的代码以下载模拟出的数据.

```
rungeKutta.downloadHistory(0, 30);
```

将 `0` 和 `30` 替换为你想要下载的被模拟出的数据的时间的区间端点. 省略这两个参数将会让它下载目前模拟出的所有数据.

6.2 改变默认模型的参数

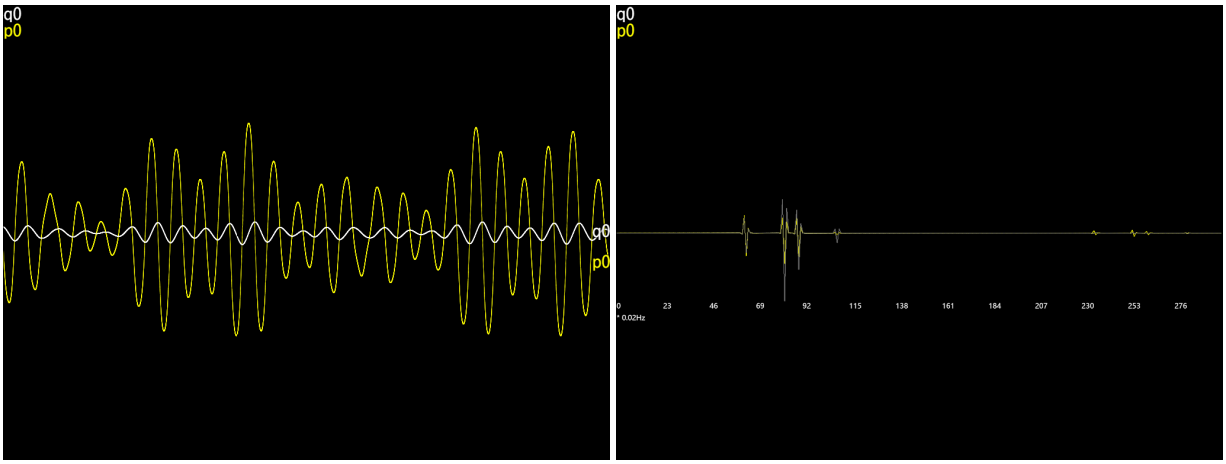
例如, 现在我们想要研究参变共振的规律. 对于一个参变振动模型, 当 $\gamma \approx 2\omega$ 时, 它达到参变共振的条件是[8] [8, p. 82]

$$|\gamma - 2\omega| < \frac{1}{2}\omega u. \tag{8}$$

我们想要研究当它达到参变共振时运动的样子. 我们可以令 $f := 0$ 来取消交变外力, 并令 $\gamma := 21.3$ 来使系统满足参变共振的条件. 然后, 运行 `restart()`.

[7]对于大多数模拟器, 按 F12 以打开控制台. 控制台的界面通常像图 5.

[8]精确到 $\mathrm{O}(u)$.



(a) 默认模型模拟出的运动　　(b) 默认模型中 p 的频域

图 6: 模拟器模拟默认模型的截屏

```
f = 0;
gamma = 21.3;
restart();
```

6.3 改变初条件

初条件 $\mathbf{x}(0)$ 可以被自定义, 通过运行下面的代码.

```
rungeKutta.initial = [2, 0];
restart();
```

将 `[2, 0]` 改为任意想要的初条件.

6.4 改变尺度

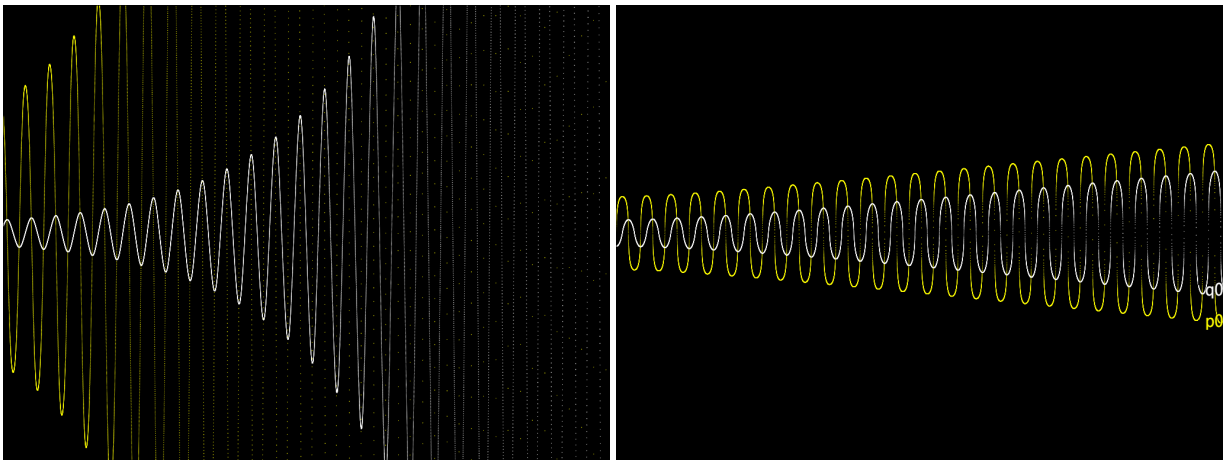
可以发现, 振幅确实如理论预言般指数增长. 因为指数增长是很快的, 图像很快超出了屏幕范围. 我们可以使用对数尺度来避免这一点. 这可以通过用下面代码改变 m_y 来完成.

```
canvas.mappingY = y => 20 * log1p(abs(y)) * sign(y) + Graphics.height/2;
restart();
```

现在 $m_y(y) := 20\ln(1+|y|)\operatorname{sgn} y + h/2$.
注意, 在代码中 `Graphics.width` 代表 w, `Graphics.height` 代表 h.
参变共振的模拟结果可以在图 7 中被看到. 可以看到振幅确实在按指数增加.

6.5 关闭频域分析

你还能通过下面的代码关闭频域分析. 运行该代码后, 左上角不会再出现表明频域已准备好的按钮.



(a) 参变共振现象被模拟　　(b) 使用对数尺度防止振幅增长过快

图 7: 模拟器模拟参变共振

```
canvas.detectPeriod = false;
restart();
```

6.6 高级: 绘制相图

相图是表示相点 **x** 的运动的图像 [8, p. 146][2, p. 68].

模拟器带有允许用户在 ODE 求解器推进时运行自定义代码的 API. 为了绘制相图, 首先你需要用下面的代码创建 `Sprite` 对象和 `Bitmap` 对象.

```
var phaseSprite = new Sprite();
phaseSprite.bitmap = new Bitmap(Graphics.width, Graphics.height);
scene.addChild(phaseSprite);
```

然后, 改变 `canvas.onTrace`, 用于在新的点被添加时运行自定义代码, 并 `restart()`.

```
canvas.onTrace = (t, qp) => {
  phaseSprite.bitmap.setPixel(...qp.map(canvas.mappingY), 'white');
  return true;
};
restart();
```

如果你想, 你可以通过运行 `canvas.visible = false;` 来隐藏原始画布.

也能在任何时候通过运行 `phaseSprite.bitmap.clear();` 来清空相图.

图 8 展示了上面的代码的结果.

每个周期的相图的边上的线应当是连续的, 但是因为相点移动得过快, 模拟器不能连续地追踪它. 这些离散的点看似在相图中形成了曲线, 可以在图 8 中看出来. 这些曲线的方程是什么? 看上去这个模拟器启发我们回答这样的问题并鼓励我们学习一些像这样的有关物理的东西.

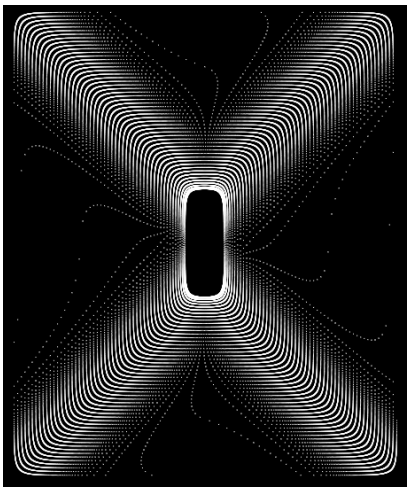


图 8: 对数尺度下参变共振运动的相图

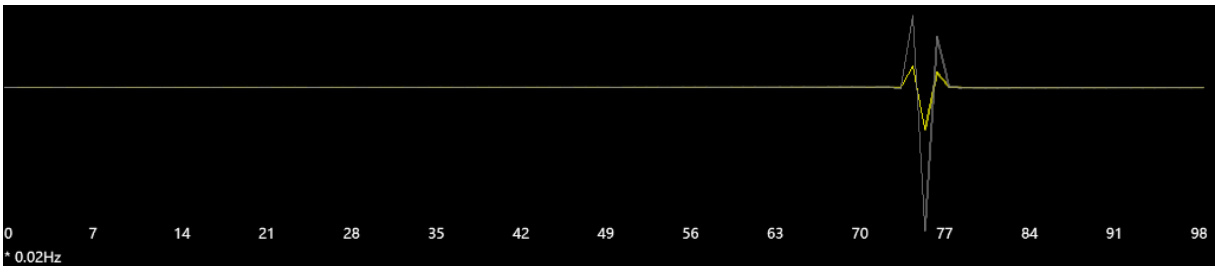


图 9: 非线性振子的频域 (高频区域没有显示)

6.7 将 Hamiltonian 改成非线性振子的

用户可以将一个函数赋值给变量 `rungeKutta.func` 来改变式 1 中的 **f**. 为了按照式 5 根据 Hamiltonian 创建 **f**, 使用函数 `canonicalEquation`. 其第一个参数为 DOF, 第二个参数为 Hamiltonian 函数.

一个例子是将默认模型的 Hamiltonian 改为非线性振子的

$$\mathcal{H}(t,q,p) := \frac{p^2}{2} + \frac{\omega_0^2 q^2}{2} + \alpha q^3 + \beta q^4$$

通过运行下面的代码. 在使用该代码前, 如果你已经运行了一些上面的代码, 刷新网页[9]已获得清洁的环境, 防止上面的更改产生效果.

```
rungeKutta.func = canonicalEquation(1, (t, qp) => {
  let [q, p] = qp;
  return p**2/2 + omega0**2*q**2/2 + alpha*q**3 + beta*q**4;
});
```

参数 $(\omega_0, \alpha, \beta)$ 需要被定义. 假设我们取

$$(\omega_0, \alpha, \beta) := (10, -2, -3)\,,$$

然后我们能运行下面的代码.

[9]对于大多数浏览器, 刷新的快捷键是 F5.

```
var omega0 = 10;
var alpha = -2;
var beta = -3;
```

如果我们通过下面的代码取初条件 $(q,p)=(1,0)$, 振幅被设定为 $b=1$.

```
rungeKutta.initial = [1, 0];
```

我们可以用模拟器验证计算非线性振子的振动频率的公式 [8, p. 87][10]

$$\omega=\omega_0+\left(\frac{3\beta}{2\omega_0}-\frac{15\alpha^2}{4\omega_0^3}\right)b^2=9.535.$$

频域在图 9 中被展示. 可以看到, 频率大约是理论预测值 $\frac{\omega}{2\pi}=1.518$, 与线性振子的频率 $\frac{\omega_0}{2\pi}=1.592$ 不同.

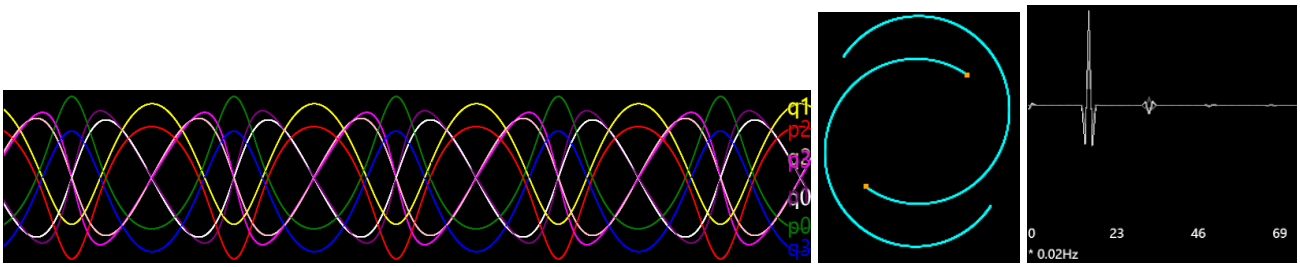
7 案例

在节 6 被呈现的默认模型和对它的典型的自定义是模拟器的好的应用案例
有一些其他的案例. 这些案例的代码可以在网页上被找到.

7.1 Kepler 二体问题

二体问题是具有 4 DOF 的系统. 将被模拟的模型的 Hamiltonian 是

$$\mathcal{H}\left(t,q_0,q_1,q_2,q_3,p_0,p_1,p_2,p_3\right):=p_0^2+p_1^2+p_2^3+p_3^2-\frac{500}{\sqrt{\left(q_0-q_2\right)^2+\left(q_1-q_3\right)^2}}.$$



(a) **x** 的图像, 共 8 条曲线　(b) 可视化　(c) q_0 的频域

图 10: 模拟器模拟 Kepler 二体问题

运行下面的代码, 然后你会得到两个互相运动的恒星.

```
// 画布上有 8 条曲线将被绘制
canvas.n = 8;
// 设置系统的 Hamiltonian
```

[10] 精确到 $\mathrm{O}\left(b^2\right)$.

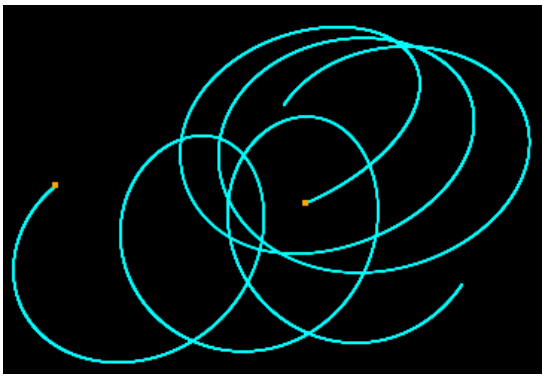
```
  rungeKutta.func = canonicalEquation(4, (t, qp) => {
    let [x1, y1, x2, y2, px1, py1, px2, py2] = qp;
    return px1**2 + py1**2 + px2**2 + py2**2 - 500 / hypot(x1-x2, y1-y2);
  });
  // 设置系统的初条件
  rungeKutta.initial = [-3, -3, 3, 3, 2, -3, -2, 3];
  // 设置图像的尺度
  canvas.mappingY = y => 20 * y + Graphics.height/2;
  // 设置图像的颜色. 有 8 条曲线, 所以设置 8 个颜色
  canvas.colors = ["white", "yellow", "pink", "blue",
                   "green", "purple", "red", "magenta"];

  // 下面的代码创建用于可视化的 Sprite 对象和 Bitmap 对象
  // 使用 rpg_core.js 提供的 API, 见 [1] 获取帮助
  var traceSprite = new Sprite();
  var star1 = new Sprite();
  var star2 = new Sprite();
  scene.addChild(traceSprite);
  scene.addChild(star1);
  scene.addChild(star2);
  traceSprite.bitmap = new Bitmap(Graphics.width, Graphics.height);
  star1.bitmap = star2.bitmap = new Bitmap(4, 4);
  star1.bitmap.fillAll('orange');
  star1.anchor.x = star1.anchor.y = 0.5;
  star2.anchor.x = star2.anchor.y = 0.5;

  // 当新的样本出现时, 更新 Sprite 对象的状态
  canvas.onTrace = (t, qp) => {
    // 设置恒星的 Sprite 对象的位置
    [star1.x, star1.y, star2.x, star2.y] =
        qp.slice(0, 4).map(canvas.mappingY);
    // 描出轨迹
    traceSprite.bitmap.setPixel(star1.x, star1.y, 'cyan');
    traceSprite.bitmap.setPixel(star2.x, star2.y, 'cyan');
    return true;
  };

  // 重新开始以应用改变
  restart();
```

被模拟出的结果在图 10 中被展示.

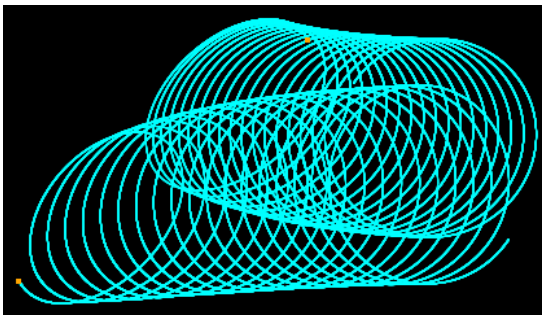
如果用户好奇, Hamiltonian 或初条件可以被修改以创建有趣的图像.

行 `rungeKutta.initial = [-3, -3, 3, 3, 5, -2, -5, 2];` 可以被修改以应用不同的初条件. 对于不同的初条件, 轨迹能呈现不同的形状. 如果我们把 `2, -3, -2, 3` 换成 `5, -2, -5, 2`, 轨迹是如图 11 所示的双曲线.

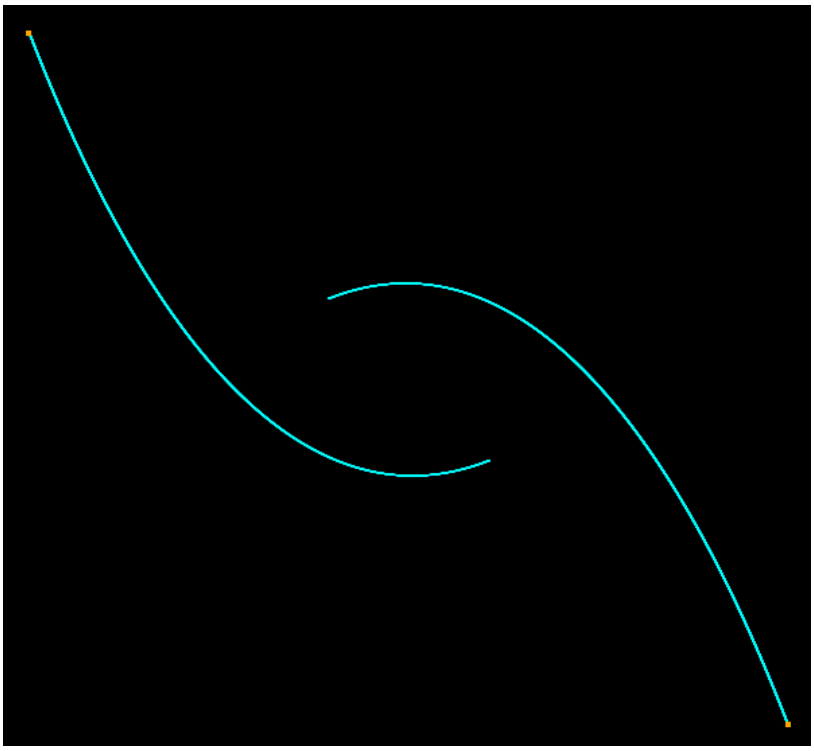
行 `return px1**2 + py1**2 + px2**2 + py2**2 - 500 / hypot(x1-x2, y1-y2);` 可以被更改以给系统应用一个不同的 Hamiltonian. 如果你将 `x1-x2` 变为 `1.5*x1-x2`, 你得到的是如图 11 所示的混乱的运动.



(a) 代码 `x1-x2` 变为 `1.5*x1-x2`



(b) 代码 `x1-x2` 变为 `1.1*x1-x2`



(c) 代码 `2, -3, -2, 3` 变为 `5, -2, -5, 2`

图 11: Kepler 二体问题模型被更改

7.2 浸渐不变量

一维周期运动的作用变量是浸渐不变量, 它在 Hamiltonian 中的参数缓慢变化时保持不变 [2, p. 298][8, p. 156].

一个与浸渐不变量有关的系统有些难以想象. 我们想要用模拟器展示浸渐不变量如何起作用. 下面展示的代码能够被修改以模拟其他与浸渐不变量有关的系统.

这里被采用的案例是频率缓慢变化的简谐振子. 它的 Hamiltonian 是

$$\mathcal{H}(t,q,p) = \frac{p^2}{2} + \omega(t)^2 \frac{q^2}{2},$$

其中 $\omega(t) := 4 + 0.05t$ 关于 t 缓慢变化.

根据作用变量的定义, 我们可以得到简谐振子的浸渐不变量为 [2, p. 300][8, p. 157]

$$I = \frac{\mathcal{H}}{\omega}.$$

我们将要让画布绘制 I 和 $\mathcal{H}$ 的图像. 运行下面的代码.

```
// 上面提到的变化参数 omega. 它应当缓慢变化
var omega = t => 4 + 0.05 * t;
// 系统的 Hamiltonian
var hamiltonian = (t, qp) => qp[1]**2/2 + omega(t)**2 * qp[0]**2/2;
// 设置系统的 Hamiltonian
rungeKutta.func = canonicalEquation(1, hamiltonian);
// 有 4 条曲线需要在画布上被绘制
canvas.n = 4;
// 曲线的标签将会是 q, p, H, I
canvas.getLabelString = i => 'qpHI'[i];
// 曲线的颜色
canvas.colors = ["white", "yellow", "pink", "blue"];
// 当新的样本产生时跟踪 I 和 H
canvas.trace = function (t, data) {
  // 计算此时的 Hamiltonian 值
  let h = hamiltonian(t, data);
  // 此时要在画布上画的内容: [q, p, h, h/omega]
  data = data.concat([h, h / omega(t)]);
  // 调用旧方法, 这是 JavaScript 的技巧
  return this.__proto__.trace.call(this, t, data);
};
// 重新开始以应用改变
restart();
```

缓变参数和 Hamiltonian 的定义被注释标记. 可以随意地改变它们来模拟其他有浸渐不变量的系统.

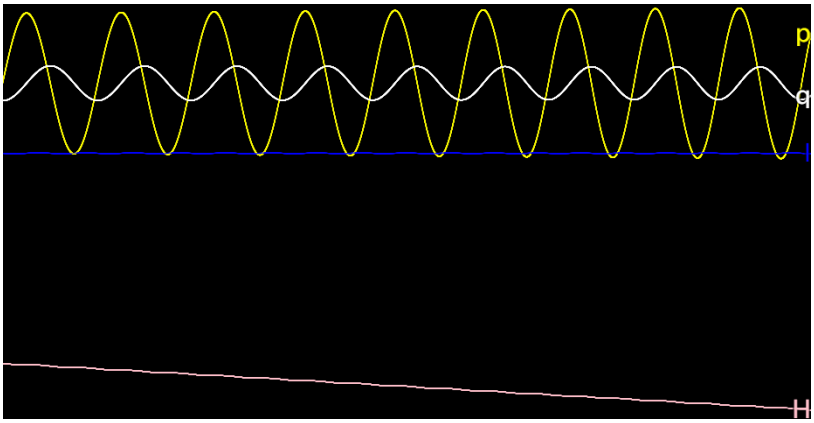


图 12: 带有缓变参数的简谐振子的浸渐不变量

图 12 展示了结果. 可以看到, 与理论预言相同, 当 ω 以及 $\mathcal{H}$ 缓变, I 不变.

7.3 粒子束散射

假设有一束全同粒子向有心力场发射, 粒子束的每个粒子具有 Hamiltonian

$$\mathcal{H}\left(t, q_0, q_1, p_0, p_1\right) = p_0^2 + p_1^2 + \frac{30}{\sqrt{q_0^2 + q_1^2}}.$$

粒子束会被散射, 具有不同的瞄准距离的粒子会有不同的散射角 [8, p. 49]. 我们想要用模拟器研究这一现象.

可以用下面的代码完成这一模拟. 注意, 在低配置设备上, 模拟是缓慢的, 因为有 30 个运动被同时模拟.

```
// 粒子束中的粒子总数
var n = 30;
// n 个 ODE 求解器的数组
rungeKuttas = [];
for (let i = 0; i < n; i++) {
  // 创建 ODE 求解器
  rungeKuttas[i] = RungeKutta.solveHamiltonian(
    // 参数列表:
    2,                              // 自由度
    [-20, (i - n/2)*0.3, 4, 0],     // 初条件
    Number.POSITIVE_INFINITY,       // 模拟的最长时间
    null,                           // 画布. null 表示无画布
    (t, qp) => {                    // Hamiltonian
      let [x, y, px, py] = qp;
      return px**2 + py**2 + 30/hypot(x,y);
    }
  );
}

// 创建用于可视化的 Sprite 对象和 Bitmap 对象 (rpg_core.js API)
var traceSprite = new Sprite();
scene.addChild(traceSprite);
traceSprite.bitmap = new Bitmap(Graphics.width, Graphics.height);

// 绘图的尺度
var my = y => 20 * y + Graphics.height/2;
// 每帧更新时调用的函数
update = function () {
  for (let i = 0; i < n; i++) {
    // 计算描点坐标
    let xy = [0, 1].map(j => my(rungeKuttas[i].current[j]))
    // 描出轨迹上的点
    traceSprite.bitmap.setPixel(...xy, 'white');
    // ODE 求解器推进
    rungeKuttas[i].update();
  }
```

```
};

// 重新开始以应用更改
restart();
// 隐藏原始画布
canvas.visible = false;
```

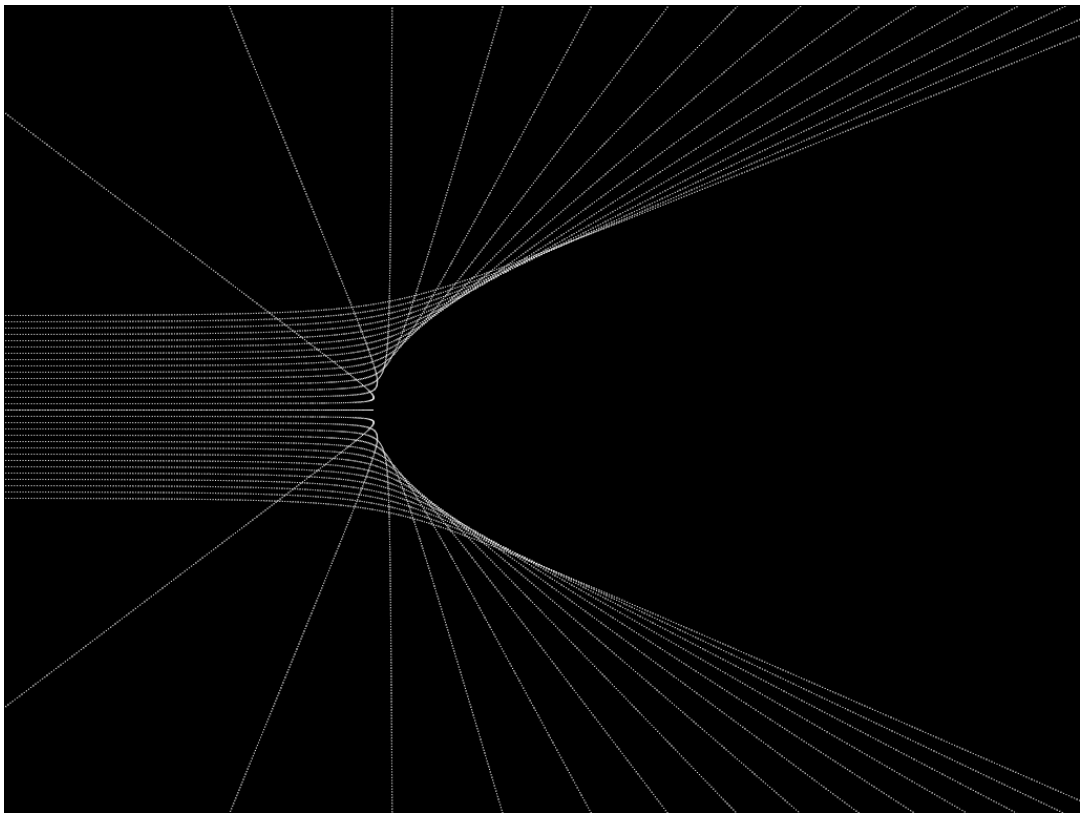


图 13: 被散射的粒子的轨迹

上面的代码的结果被图 13 展示.

散射的效应形成了优美的图像. 可以看到, 粒子的轨道的包络线看上去像是圆锥曲线, 在我们模拟它之前我们没有想到这一点. 然后, 我们可以对包络线的方程是什么感到好奇, 因为它是有意义的, 它表征了粒子无法到达的区域.

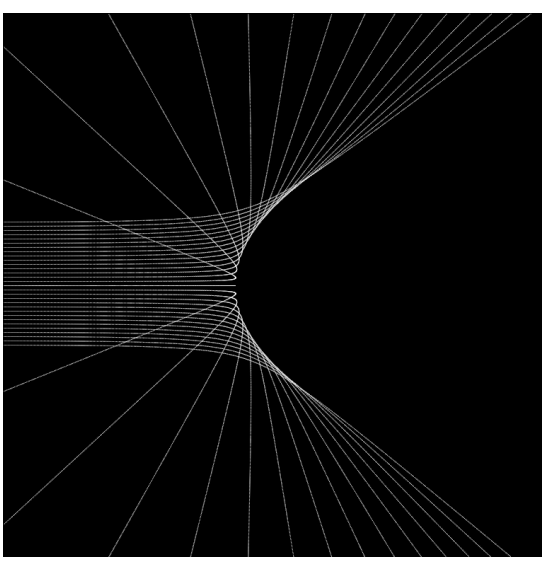
像这样简单的模拟启发我们研究物理. 这正是模拟器的一个主要用途.

人们还能研究粒子束的初始速度和场强对轨迹的影响. 我们可以用模拟器验证, 当初始速度减小或场强增大时, 轨迹会更加弯曲.

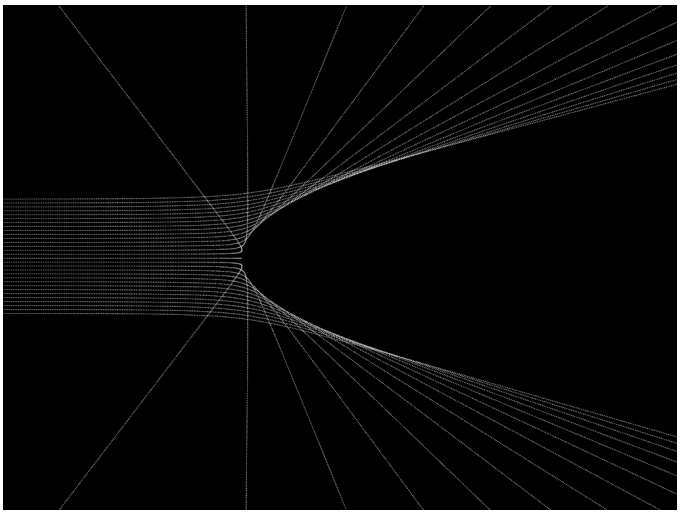
行 `[-20, (i - n/2)*0.3, 4, 0]`, 指定了第 `i` 个粒子的初条件.

行 `return px**2 + py**2 + 30/hypot(x,y);` 指定了粒子的 Hamiltonian.

我们将代码 `4, 0` 变为 `3, 0` 以使得轨迹更加弯曲, 将代码 `30/hypot(x,y)` 变为 `20/hypot(x,y)` 以使得轨迹更加平直, 见图 14.



(a) 代码 `4, 0` 变为 `3, 0`, 轨迹更加弯曲



(b) 代码 `30/hypot(x,y)` 变为 `20/hypot(x,y)`, 轨迹更加平直

图 14: 粒子束散射模型的参数被改变

7.4 狭义相对论

不仅仅是经典力学, 模拟器还能模拟狭义相对论力学, 因为相对论力学能用 Hamilton 力学描绘. 考虑相对论粒子在均匀重力场中的运动, 它具有 Hamiltonian[7, p. 28]

$$\mathcal{H}(t,q,p) := \sqrt{p^2+10} - 0.8q$$

和初条件 $(q,p) = (-10,-10)$.

模拟的结果被图 15 展示. 随着 $t \to \infty$, 该粒子动量匀速增大, 速度越来越接近光速, 符合理论预言 [7, p. 24].

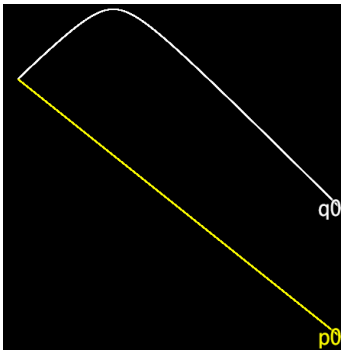


图 15: 均匀重力场中相对论粒子的运动

8 结论

我们开发了一个方便的线上软件, 它可以模拟 Hamilton 力学. 有很多能用它完成的应用案例.

模拟器是力学模拟器. 当用户输入系统的 Hamiltonian 和初条件后, 模拟器能模拟系统的运动, 在屏幕上作出运动的图像. 如果振动系统被模拟, 在足够多的样本被计算出来后, 模拟器还能用 FFT 计算出运动的频域.

模拟器具有下面的优点:

1. 体积小, 速度快. 模拟器简单但高效. 有良好网络环境的电脑的用户可以在没有准备的情况下在几秒内开始模拟力学系统.

2. 非常方便. 任何有浏览器和互联网的人都可以在不需要下载程序文件到磁盘的情况下使用模拟器. 模拟器基于 HTML, 程序用 JavaScript 写成, 被几乎所有浏览器支持.

3. 高度可自定义. 模拟器的一切都可以被容易地自定义. 被模拟的模型, ODE 求解器的参数, 模拟器呈现系统的方式, 等等都能被自定义.

4. 可以输出数据. 被模拟出的数据可以被输出, 用于创建力学数据集. 被创建的数据集可以被用于研究系统的规律或者被第三方软件分析.

5. 容易操作. 用户需要做的一切就是在控制台中编写简单的 JavaScript 代码和点击屏幕. 编写代码是简单的. 如果不需要高级用法, 没有编程经历的用户只需要几分钟时间学会使用它.

模拟器可以被用于研究 Hamilton 力学系统的运动, 学习经典理论力学, 为与物理相关的讲座或者物理课程创建动态图片, 和创建 Hamilton 系统的运动的数据集.

我们写了默认模型, 用于说明它的基础应用, 在节 6 中被提到. 在节 7 中, 我们还提到了一些其他的案例, 用于解释一些更深远的应用.

它被挂载在网页上.

参考文献

[1] Rpg maker mv help.

[2] V. I. Arnol′d, K. Vogtmann, and A. Weinstein. *Mathematical methods of classical mechanics*. Springer, 2nd edition, 1989.

[3] K. Basques. Console overview, 2020.

[4] E. Hairer, S. P. Nørsett, and G. Wanner. *Solving ordinary differential equations I: Nonstiff problems*. Springer, 2008.

[5] L. N. Hand and J. D. Finch. *Analytical mechanics*. Cambridge University Press, 2008.

[6] F. J. Harris. On the use of windows for harmonic analysis with the discrete fourier transform. *Proceedings of the IEEE*, 66(1):51–83, 1978.

[7] L. D. Landau and L. E. M. *The classical theory of fields*. Butterworth Heinemann, 2010.

[8] L. D. Landau, L. E. Mikhaĭlovich, J. B. Sykes, and J. S. Bell. *Mechanics*. Butterworth-Heinemann, 3rd edition, 1976.

[9] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical recipes: the art of scientific computing*. Cambridge University Press, 2007.

致谢

该课题源自我对振动的研究. 我需要编写一个方便的力学模拟器来研究它的运动. 我想要感谢我的导师李晟, 因为在该过程中, 他提供给我一些有关模拟器的功能的想法, 比如 FFT 分析.

我想要感谢我的父母, 他们在我在研究中止步不前时提供给我精神支持.