桥梁移动车辆的实时监测生成及分析预警系统

摘 要

如今，中国运营中桥梁的保有量巨大。为确保公路交通系统的安全有效运行，对桥梁结构健康进行准确、有效、实时监测的需求日益增长。桥梁结构健康监测系统已成为了路桥行业中的重要领域。

本研究尝试了“桥梁移动车辆的实时监测生成及分析预警系统”在车辆荷载作用方面的实现，即主动获取桥梁结构的实时车辆荷载，在车辆荷载上桥前预先获取相关的参数信息、通过后台计算提前分析行驶车辆对桥梁结构的激励效应，并对可能危害桥梁运营安全的车辆荷载进行及时预警。本研究通过模块化的表达，完成了主动化桥梁结构健康监测系统的工作流程阐述。采用 Python 语言编写程序代码，以面向对象的程序设计方法实现了匀速移动车辆荷载的实时生成、数据流传输、荷载管理及动态作用模拟。通过荷载数据接口完成了桥梁结构在移动车辆荷载作用下的响应实时运算，通过对桥梁响应数据池共享，实现了对结构安全的评估预警，并利用多线程技术实现了上述功能模块的实时协同实现。通过对一简支梁模型进行算例应用，初步验证了桥梁移动车辆的实时监测生成及分析预警系统的可行性和适用性。

关键词：移动车辆荷载，桥梁，健康监测，预警，实时操作系统，多线程协同

A simulation program of a real-time vehicle loads monitoring and early warning system

ABSTRACT

Nowadays a large amount of bridges are operating widely in China. In order to ensure the performance and efficiency of the nation-wide highway transportation system, the demand of an accurate and real-time monitoring system for bridges' health is increasingly rising. The Structural Health Monitoring System of bridges has become a considerable researching and engineering area in the bridge industry.

The Structural Health Real-time Vehicle Load Monitoring and Early Warning System was brought forward in the aspect of vehicle load effect, as to acquire the real-time information of vehicle loads in advance, calculate and predict the structural effect in the background in advance, and make warning to the potential dangers in time. The working process of the system was explained by modules in detail. A real-time simulation program generating vehicle loads with constant velocity, transporting the data flow, managing vehicle loads and calculating the real-time structural dynamic effect, was made by Python using the Object Oriented Programming method. The real-time dynamic data flow of vehicle loads and bridge effects was realized with the data interfaces of loads. The safety assessment and in-time alert was available by using shared data pools of effect. All function modules listed were gathered to run under a real-time multi-threaded coordination. The concept of the system was primarily verified to be feasible and serviceable with an application onto a 50-meter simple supported girder bridge.

Key words: Vehicle Load, Bridge, Health Monitoring, Early Warning, Real-time Operating System (RTOS), Multi-threaded Coordination

目 录

1 绪 论 1
1.1 研究背景 1
1.2 研究构想 2
1.3 研究内容 3
2 基本理论 4
2.1 桥梁结构振动 4
2.2 有限单元法 12
3 荷载生成与效应分析 15
3.1 车辆荷载参数 15
3.2 荷载生成的一般考虑 16
3.3 效应分析 17
4 面向对象的 Python 程序开发 18
4.1 Python 语言 18
4.2 程序结构 18
5 研究结果 23
5.1 模型参数及生成 23
5.2 程序代码 25
5.3 程序效果演示 25
6 结论与展望 29
6.1 结论 29
6.2 展望 29
7 致谢 32
8 附录 程序代码 33

1 绪 论

1.1 研究背景

1.1.1 桥梁结构健康监测系统

结构健康监测系统（Structural Health Monitoring System，简称 SHMS）诞生于 20 世纪末。随着材料、传感器、计算机、通讯、数据分析等技术的发展，它在近 20 年中很快成为了全世界的研究热点，得到了大量讨论与应用。

目前，有人将 SHMS 定义为“通过对包括结构响应在内的结构特性的现场无损传感和分析,达到侦测结构损伤或退化目的的技术”[1]。也有人将 SHMS 定义为“通过一系列传感设备,也可能包括作动设备,以将无损检测技术嵌入结构和材料的方式对结构荷载、损伤状态进行记录、分析、定位和预测的技术”[2]。由上述两种定义，SHMS 作为以侦测、预测桥梁结构荷载与损伤状态为目的的技术，其根本目的关键在于达到对运营状态的“实时评估”。

随着中国公路、铁道路网保持着高速度的建设，人们可以预见，人国未来的新建桥梁建设需求会缓慢趋于饱和。面对趋于饱和的市场，未来的整体工程设计将呈“精细化设计”的大方向。同样地，结构健康监测领域也有逐渐走向“精细化监测”的趋势。同时，在新建桥梁数量逐步降低的时代背景下，人国运营中桥梁的数量会保持在较高水平。据不完全统计，截止至 2012 年，中国建成运营中的公路铁路桥梁总数已达到 50 余万座[3]。纵观全球，面对数量巨大的已建桥梁，桥梁运营状态下的健康监测、预警将成为一个庞大并且重要的市场。在如此庞大的市场需求下，推动 SHMS 的发展，并更加有效地管理、养护数量众多的桥梁设施，已成为业主、养护单位、科研单位、政府部门与社会公众的共同要求[4]。

1.1.2 发展现状

国际上，1987 年，英国在 Foyle 大桥上布置传感器，监测大桥运营时在车辆以及风荷载作用下的振动、挠度以及应变响应，是较为完整的桥梁结构健康监测系统的最早应用之一[5]。加拿大在 Confederation 大桥上实施了桥梁结构健康监测系统，主要针对桥梁在冰荷载作用下结构的长期变形、桥梁在车辆荷载下的动力响应、环境对桥梁的侵蚀作用进行了监测[6]。在中国国内，上海徐浦大桥安装了桥梁结构健康监测系统，具有监测车辆荷载的功能，并监测中跨主梁标高、温度、应变，监测拉索应力[7]。贯通香港澳门珠海两岸三地的港珠澳大桥进行了较为深入的结构健康监测系统总体设计，具有电子化人工巡检维护、监测评估结构构件风险与损坏、进行养护决策的功能。然而目前，尚没有投入实际工程使用的、对桥梁移动车辆荷载进行实时分析的预警功能。

1.1.3 存在的问题

在桥梁结构健康监测系统的实际应用中，车辆荷载对桥梁结构的影响是一个非常重要的

因素。根据实际车流状况实时生成的车辆荷载可为桥梁结构提供接近真实的车辆荷载作用形式，对于在线分析评估桥梁结构安全性具有重要意义。但是目前，利用有限元软件对车辆荷载影响的分析还不能做到实时进行。主要原因在于，传统的有限元分析软件进行桥梁车载计算时，需要预先把所有荷载信息编成数据卡片，然后输入软件进行计算，获得结果后就完成了任务。在这个过程中，往往是导入一个由历史车辆荷载监测数据包所建立的交通荷载模型，将这个单一的荷载模型作为有限元计算的车辆荷载。如此完成的监测是被动的、滞后的。这种荷载生成方式不能满足 SHMS 中实时进行分析的要求，无法达到精细化监测的高度，更不能实现及时、提前预警。在这样的背景条件下，建立一个实现实时导入车辆荷载并实时进行计算分析的监测、预警体系，具有实际的工程意义。

1.2 研究构想

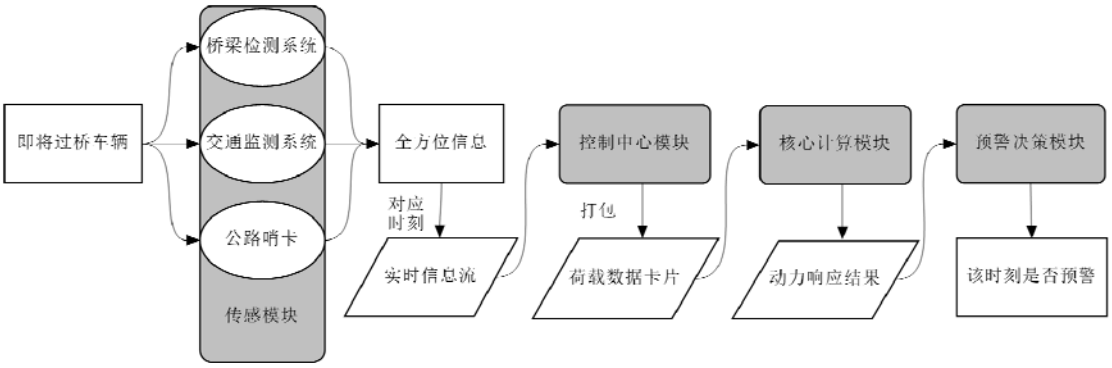


图 2.1 构想中的分析流程图

- 利用现有的桥梁监测系统、公路哨卡、交通检测系统，主动获得所有即将通过指定桥梁的车辆的全方位信息。具体来讲，本次研究设想初步以一个主跨 50 米预应力混凝土简支梁桥作为监控模拟对象，其主跨的两端分别设置 50 米的引桥，在两引桥外端部的位置获取荷载的全部信息；
- 由系统将各传感器所采集的车辆荷载信息，以实时信息流的形式直接输送到“控制中心模块”。这时车辆已在引桥上，而未上主桥。利用车辆在引桥端部的时间信息、速度、引桥长度、主桥长度，可以预先算出车辆行驶至主桥的预计时间、行驶离主桥的预计时间；
- “控制中心模块”将获取的这些车辆荷载数据进行整理、打包为每单位时间的数据卡片，输出传送给“核心计算模块”；
- 在上桥预计时间、离桥预计时间的时间区域内，提前于车辆荷载实际上桥，“核心计算模块”在毫秒级“时间控制模块”下进行动力分析，得出计算响应结果，输出给“预警决策模块”以及“图形绘制模块”；
- “预警决策模块”基于计算结果进行决策，判断对车辆荷载是否进行干预。如此，即在车辆荷载实际上桥前，完成实时、主动、精细化的 SHM 预警。

1.3 研究内容

- 总结整理通过各种传感器与监测手段可获得的车辆荷载参数，包括且不限于：轴距、轴重、速度、位置、运动方向、轨迹、上桥时刻、离桥时刻。
- 模拟实际监测设备的数据生成方式，实现实时车辆荷载参数的数据生成、打包、传输的程序化模块。
- 对动力效应计算方法进行讨论与推导。
- 组织模块间同步的多线程并发运算，完成代码，调试程序，实现模拟。

2 基本理论

2.1 桥梁结构振动

2.1.1 单自由度振子模态分析[8]

以带有均布静荷载的等截面简支梁桥为模型，其模态分析如下：

静力弹性平衡的微分方程为：

$$EIy_0^{IV} = q \tag{2.1.1}$$

.

假设围绕静挠度 y_0 发生振幅为 y 的固有振动，根据达朗贝尔原理，得到动力弹性平衡的微分方程为：

$$EI\left(y_0 + y\right)^{IV} = q - m\ddot{y} \tag{2.1.2}$$

将式(2.1.2)减去式(2.1.1)得：

$$EIy^{IV} + m\ddot{y} = 0 \tag{2.1.3}$$

式(2.1.3)是一个常系数非齐次线性微分方程，可以使用分离变量法得：

$$y\left(\mathrm{x},t\right) = Y\left(x\right)\cdot F\left(t\right) \tag{2.1.4}$$

其中 $Y\left(x\right)$ 为振型，振型是系统固有的，与时间无关。将式(4.4)代入式(4.3)，得到变量分离为：

$$\frac{EIY^{IV}}{mY} = -\frac{\ddot{F}}{F} = \omega_n^{\,2} \tag{2.1.5}$$

其中 ω 为常数，即简支梁桥的固有频率。下面以 ω 作为未知数求解，由式(4.5)分离为两个独立的微分方程：

$$\ddot{F} + \omega_n^{\,2}F = 0 \tag{2.1.6}$$

$$EIY^{IV} - m\omega_n^{\,2}Y = 0 \tag{2.1.7}$$

对式(2.1.6)求解：

$$F\left(t\right) = M\sin\omega_n t + N\cos\omega_n t = \alpha\sin\left(\omega_n t + \nu\right) \tag{2.1.8}$$

其中 α 为振幅，ν 为初相位。

对式(2.1.7)求解：

$$Y(x)=A\cosh kx+B\sinh kx+C\cos kx+D\sin kx \tag{2.1.9}$$

其中 $k=\sqrt[4]{\dfrac{\omega_n{}^2 m}{EI}}$ 。

对于跨度为 l 的简支梁桥有边界条件：

$$Y(l)=Y''(l)=0 \tag{2.1.10}$$

对式(2.1.9)的两侧求二阶偏导：

$$Y''(x)=Ak^2\cosh kx+Bk^2\sinh kx-Ck^2\cos kx-Dk^2\sin kx \tag{2.1.11}$$

将式(2.1.10)的初始条件代入式(2.1.11)中：

$$\because B\sinh kl+D\sin kl=B\sinh kl-D\sin kl=0$$
$$\therefore \begin{cases}2B\sinh kl=0\\2D\sin kl=0\end{cases}$$
$$\because \forall \sinh kl\neq 0$$
$$\therefore B=0$$
$$\because Y(z)\neq 0\wedge A=B=C=0$$
$$\therefore D\neq 0\wedge \sin kl=0$$
$$\therefore kl=n\pi,(n=1,2,3\cdots)$$

则式(2.1.9)可以简化为：

$$Y_n(x)=D_n\sin\frac{n\pi x}{l},(n=1,2,3\cdots) \tag{2.1.12}$$

将式(2.1.12)代入式(2.1.7)求 ω_n 为：

$$Y_n^{IV}(x)=D_n\left(\frac{n\pi}{l}\right)^4\sin\frac{n\pi x}{l}$$
$$\because EID_n\left(\frac{n\pi}{l}\right)^4\sin\frac{n\pi x}{l}-m\omega_n{}^2D_n\sin\frac{n\pi x}{l}=0$$
$$\therefore \omega_n{}^2=\left(\frac{n\pi}{l}\right)^4\frac{EI}{m}$$

得到简支梁桥的 n 阶固有频率为：

$$\omega_n = \left(\frac{n\pi}{l}\right)^2 \sqrt{\frac{EI}{m}} \tag{2.1.13}$$

2.1.2 单自由度振子有阻尼简谐荷载受迫振动分析

此推导系参考李国豪所著《桥梁结构稳定与振动》[8]中第 292 页对无阻尼的简支梁在匀速移动集中力荷载下的振动响应推导，进一步地完成了有阻尼的简支梁桥在简谐激励作用下的振动响应推导分析。步骤如下：

匀速移动常量集中力作用下，有阻尼的简支梁桥受迫振动方程：

$$EI\frac{\partial^4 y}{\partial x^4} + m\frac{\partial^2 y}{\partial t^2} + c\frac{\partial y}{\partial t} = F(x,t) \tag{2.2.1}$$

其中，EI 为主梁刚度，m 为主梁质量，c 为粘滞阻尼系数。

将简谐激励荷载表示为 $F(x,t) = F\cdot\delta(x-vt)$，它等同于荷载大小为 F 的匀速移动集中力作用在桥上期间的作用激励。

由式（2.1.12），对于简支梁桥，$\phi_n = \sin\frac{n\pi x}{l}$ 。

将动力位移 $y(x,t)$ 分解为振型的级数形式：

$$y(x,t) = \sum_{n=1}^{N} A_n(t)\phi_n(x) \tag{2.2.2}$$

将式(2.2.2)代入(2.2.1)得：

$$EI\sum_{n=1}^{N} A_n(t)\left(\frac{n\pi}{l}\right)^4 \sin\frac{n\pi x}{l} + m\sum_{n=1}^{N} \ddot{A}_n(t)\sin\frac{n\pi x}{l} + c\sum_{n=1}^{N} \dot{A}_n(t)\sin\frac{n\pi x}{l} = F(x,t) \tag{2.2.3}$$

令 $F(x,t) = \sum_{n=1}^{N} B_n(t)\sin\frac{n\pi x}{l}$ ，利用振型的正交性，则将式(2.2.3)解耦可得：

$$EIA_n(t)\left(\frac{n\pi}{l}\right)^4+m\ddot{A}_n(t)+c\dot{A}_n(t)=B_n(t) \tag{2.2.4}$$

可将 $B_n(t)$ 展开为：

$$\begin{aligned}B_n(t) &= \frac{\int_0^l F(x,t)\cdot\phi_n(x)dx}{\int_0^l \phi_n{}^2(x)dx}\\ &= \frac{\int_0^l F\cdot\delta(x-vt)\cdot\sin\frac{n\pi x}{l}dx}{\int_0^l \sin^2\frac{n\pi x}{l}dx}\\ &= \frac{F\cdot\sin\frac{n\pi vt}{l}}{l/2}\\ &= \frac{2F}{l}\cdot\sin\frac{n\pi vt}{l}\end{aligned}$$

令 $\frac{c}{m}=2\xi\omega_n$ 。

其中 ξ 为阻尼比，且 $\omega_n=\left(\frac{n\pi}{l}\right)^2\sqrt{\frac{EI}{m}}$ 为 n 阶固有频率。

则，式(2.2.4)可表示为：

$$\ddot{A}_n(t)+2\xi\omega_n\dot{A}_n(t)+\omega_n{}^2A_n(t)=\frac{2F}{ml}\cdot\sin\frac{n\pi vt}{l} \tag{2.2.5}$$

令 $\omega=\frac{n\pi v}{l}$ ， $p_0=\frac{2F}{l}$，将式(2.2.4)简化为：

$$\ddot{A}_n(t)+2\xi\omega_n\dot{A}_n(t)+\omega_n{}^2A_n(t)=\frac{p_0}{m}\cdot\sin\omega t \tag{2.2.6}$$

式(2.2.6)的特解有形式 $A_n(t)=C\sin\omega t+D\cos\omega t$ ，求其一阶及二阶偏导：

$$\begin{aligned}\dot{A}_n(t)&=C\omega\cos\omega t-D\omega\sin\omega t\\ \ddot{A}_n(t)&=-C\omega^2\sin\omega t-D\omega^2\cos\omega t\end{aligned}$$

分别代入式(2.2.6)得：

$$\forall t\in\left(0,\frac{l}{v}\right),$$
$$\left[C\left(\omega_n{}^2-\omega^2\right)-2\xi\omega_n\omega D\right]\sin\omega t+\left[2\xi\omega_n\omega C+D\left(\omega_n{}^2-\omega^2\right)\right]\cos\omega t=\frac{p_0}{m}\sin\omega t \tag{2.2.7}$$

将正弦项与余弦项系数分离，得：

$$\begin{cases}C\left(\omega_n{}^2-\omega^2\right)-2\xi\omega_n\omega D=\dfrac{p_0}{m}\\ 2\xi\omega_n\omega C+D\left(\omega_n{}^2-\omega^2\right)=0\end{cases} \tag{2.2.8}$$

以 C,D 作为未知数，求解方程组(2.2.8)得：

$$C=\frac{p_0}{m\omega_n{}^2}\frac{1-\left(\omega/\omega_n\right)^2}{\left[1-\left(\omega/\omega_n\right)^2\right]^2+\left[2\xi\left(\omega/\omega_n\right)\right]^2}$$
$$D=\frac{p_0}{m\omega_n{}^2}\frac{-2\xi\left(\omega/\omega_n\right)}{\left[1-\left(\omega/\omega_n\right)^2\right]^2+\left[2\xi\left(\omega/\omega_n\right)\right]^2}$$

式(2.2.6)的余解为自由振动反应部分。有阻尼单自由度体系自由振动微分方程为：

$$\ddot{A}_{cn}(t)+2\xi\omega_n\dot{A}_{cn}(t)+\omega_n{}^2A_{cn}(t)=0 \tag{2.2.9}$$

桥梁中的阻尼属于欠阻尼体系，即 $\xi<1$ 。大多数结构阻尼范围为 $\xi\in\left(0,0.2\right)$ 。

求解式(2.29)的二阶非齐次线性微分方程形式为：

$$A_{cn}\left(t\right)=e^{-\xi\omega_n t}\left(A\cos\omega_D t+B\sin\omega_D t\right) \tag{2.2.10}$$

其中 $\omega_D=\omega_n\sqrt{1-\xi^2}$ ， A,B 为待定系数 。

将余解与特解相加，得到式(2.2.6)的全解形式为：

$$A_n(t)=e^{-\xi\omega_n t}\left(A\cos\omega_D t+B\sin\omega_D t\right)+C\sin\omega t+D\cos\omega t \tag{2.2.11}$$

对式(2.2.11)求一次偏导：

$$\begin{aligned}\dot{A}_n\left(t\right)=&-\xi\omega_n e^{-\xi\omega_n t}\left(A\cos\omega_D t+B\sin\omega_D t\right)+\omega_D e^{-\xi\omega_n t}\left(-A\sin\omega_D t+B\cos\omega_D t\right)\\&+C\omega\cos\omega t-D\omega\sin\omega\mathrm{t}\end{aligned} \tag{2.2.12}$$

将 $t_0=0$ 作为初始条件，联立式(2.2.11)与式(2.2.12)解 A,B 得：

$$A = A_n(0) - D$$
$$B = \dot{A}_n(0) - \xi\omega_n\left[A_n(0) - D\right]$$

故式(2.2.6)的全解为：

$$A_n(t) = e^{-\xi\omega_n t}\left(A\cos\omega_D t + B\sin\omega_D t\right) + C\sin\omega t + D\cos\omega t$$

$$\begin{cases} A = A_n(0) - \dfrac{p_0}{m\omega_n^{\,2}}\dfrac{-2\xi(\omega/\omega_n)}{\left[1-(\omega/\omega_n)^2\right]^2 + \left[2\xi(\omega/\omega_n)\right]^2} \\ B = \dot{A}_n(0) - \xi\omega_n\left[A_n(0) - \dfrac{p_0}{m\omega_n^{\,2}}\dfrac{-2\xi(\omega/\omega_n)}{\left[1-(\omega/\omega_n)^2\right]^2 + \left[2\xi(\omega/\omega_n)\right]^2}\right] \\ C = \dfrac{p_0}{m\omega_n^{\,2}}\dfrac{1-(\omega/\omega_n)^2}{\left[1-(\omega/\omega_n)^2\right]^2 + \left[2\xi(\omega/\omega_n)\right]^2} \\ D = \dfrac{p_0}{m\omega_n^{\,2}}\dfrac{-2\xi(\omega/\omega_n)}{\left[1-(\omega/\omega_n)^2\right]^2 + \left[2\xi(\omega/\omega_n)\right]^2} \end{cases} \tag{2.2.13}$$

亦可采用公式解法，即，已知有形如一般方程：

$$\begin{cases} \ddot{y} + 2\xi_n\omega_n\dot{y} + \omega_n^2 y = F_0\sin\omega(t-t_0) \\ y(t_0) = y_0, \dot{y}(t_0) = \dot{y}_0 \end{cases} \tag{2.2.14}$$

知其完全解为：

$$\begin{aligned} y(t) = {} & e^{-\xi_n\omega_n(t-t_0)}\left[y_0\cos\omega_d(t-t_0) + \frac{\dot{y}_0 + \xi_n\omega_n y_0}{\omega_d}\sin\omega_d(t-t_0)\right] \\ & + \frac{F_0\beta}{\omega_n^2}e^{-\xi_n\omega_n(t-t_0)}\left[\sin\theta\cos\omega_d(t-t_0) + \frac{\omega_n}{\omega_d}(\xi_n\sin\theta - s\cos\theta)\sin\omega_d(t-t_0)\right] \\ & + \frac{F_0\beta}{\omega_n^2}\sin\left(\omega(t-t_0) - \theta\right) \end{aligned} \tag{2.2.15}$$

其中：$\omega_d = \omega_n\sqrt{1-\xi_n^2}$，$s = \dfrac{\omega}{\omega_n}$，$\beta = \dfrac{1}{\sqrt{(1-s^2)^2 + (2\xi_n s)^2}}$，$\theta = tg^{-1}\dfrac{2\xi_n s}{1-s^2}$

则，代入 $F_0=2F/ml,\omega=n\pi v/l,y_0=0,\dot{y}_0=0$ ，得到上式解为：

$$A_n(t)=\frac{2F\beta}{ml\omega_n^2}e^{-\xi_n\omega_n(t-t_0)}\left[\sin\theta\cos\omega_d(t-t_0)+\frac{\omega_n}{\omega_d}(\xi_n\sin\theta-s\cos\theta)\sin\omega_d(t-t_0)\right]+\frac{2F\beta}{ml\omega_n^2}\sin\left(\omega(t-t_0)-\theta\right) \tag{2.2.16}$$

其中：

$\omega_d=\omega_n\sqrt{1-\xi_n^2}$ ， $\omega=\frac{n\pi v}{l}$ ， $s=\frac{\omega}{\omega_n}$ ， $\beta=\frac{1}{\sqrt{(1-s^2)^2+(2\xi_n s)^2}}$ ， $\theta=tg^{-1}\frac{2\xi_n s}{1-s^2}$

将上式简化为：

$$A_n(t)=Me^{-\xi_n\omega_n(t-t_0)}\sin\left[\omega_d(t-t_0)+\varphi\right]+N\sin\left(\omega(t-t_0)-\theta\right) \tag{2.2.17}$$

其中：

$M=\frac{2F\beta}{ml\omega_n^2}\sqrt{\sin^2\theta+\frac{\omega_n^2}{\omega_d^2}(\xi_n\sin\theta-s\cos\theta)^2}$ ， $\varphi=\arctan\frac{\sin\theta}{\frac{\omega_n}{\omega_d}(\xi_n\sin\theta-s\cos\theta)}$ ，

$N=\frac{2F\beta}{ml\omega_n^2}$

对上式求导，可得 $A_n(t)$ 的一阶导数为

$$\dot{A}_n(t)=M\omega_n e^{-\xi_n\omega_n(t-t_0)}\sin\left[\omega_d(t-t_0)+\varphi+\psi\right]+N\omega\cos\left(\omega(t-t_0)-\theta\right) \tag{2.2.18}$$

其中：$\psi=\pi-\arcsin\sqrt{1-\xi_n^2}$

则其终止时刻（荷载恰好离开桥梁时刻）的位移与速度为

$$A_n(t_0+T)=Me^{-\xi_n\omega_n T}\sin\left(\omega_d T+\varphi\right)+N\sin\left(\omega T-\theta\right) \tag{2.2.19}$$

$$\dot{A}_n(t_0+T)=M\omega_n e^{-\xi_n\omega_n T}\sin\left(\omega_d T+\varphi+\psi\right)+N\omega\cos\left(\omega T-\theta\right) \tag{2.2.20}$$

其中 T 为荷载作用在桥面的时程。

2.1.3 单自由度振子有阻尼衰减自由振动分析[9]

在无荷载作用时，单自由度振子做有阻尼衰减自由振动，其方程为：

$$\ddot{A}_n(t)+2\xi_n\omega_n\dot{A}_n(t)+\omega_n^2A_n(t)=0$$

(2.3.1)

以（2.1.2）中终止时刻的位移与速度为初始条件：$A_T=A_n(t_0+T),\dot{A}_T=\dot{A}_n(t_0+T)$

即：

$$A_n(t_0+T)=Me^{-\xi_n\omega_nT}\sin(\omega_dT+\varphi)+N\sin(\omega T-\theta)=P$$

$$\dot{A}_n(t_0+T)=M\omega_ne^{-\xi_n\omega_nT}\sin(\omega_dT+\varphi+\psi)+N\omega\cos(\omega T-\theta)=Q$$

代入 $F_0=0,\omega=n\pi v/l,y_0=P,\dot{y}_0=Q$ 到式（2.3.15），有其解为

$$A_n(t)=e^{-\xi_n\omega_n(t-t_0-T)}\left[P\cos\omega_d(t-t_0-T)+\frac{Q+\xi_n\omega_nP}{\omega_d}\sin\omega_d(t-t_0-T)\right] \quad (2.3.2)$$

2.1.4 匀速移动集中力对简支梁桥各阶模态振型的响应分析

综上所述，由于匀速移动集中力作用在简支梁桥上的时间是有限的，为 T，故对于匀速移动集中力的第 n 阶模态的广义坐标为分段函数：

$$A_n(t)=\begin{cases}0 & t-t_0<0\\ Me^{-\xi_n\omega_n(t-t_0)}\sin\left[\omega_d(t-t_0)+\varphi\right]+N\sin\left(\omega(t-t_0)-\theta\right) & 0\le t-t_0\le T\\ e^{-\xi_n\omega_n(t-t_0-T)}\left[P\cos\omega_d(t-t_0-T)+\dfrac{Q+\xi_n\omega_nP}{\omega_d}\sin\omega_d(t-t_0-T)\right] & t-t_0>T\end{cases}$$

简支梁桥在移动荷载作用下的响应即为：

$$y(x,t)=\sum_{n=1}^{N}A_n(t)\sin\frac{n\pi x}{l}$$

(2.4.1)

2.2 有限单元法

2.2.1 概述

当面对更加复杂的桥型时，往往无法顺利得到解析解，这时，就需要采用有限单元法求解数值解。基于本研究的概念之上，使用通用的信息接口，将基于经典理论的核心计算模块更换为适应性更为广泛的有限单元法计算模块，是理论上可行并有效的。

有限单元法是目前的工程分析领域应用最为广泛的数值计算方法。它借助当今飞速发展的计算机技术，得益于其自身卓越的通用性与有效性，成功大规模应用于工程计算中。

为了解决工程领域中的力学与物理问题，人们将这些问题表达为数学模型，一般为微分方程的形式，并且加以一定的定解条件。为了求解这些微分方程，人们在探索中逐步得到了许多数值求解方法。求解方法主要分为两大类：有限差分法、有限单元法。这两大类方法的主要差别在于，有限差分法的求解方程建立于固结于空间的欧拉坐标系，而有限单元法的求解方程建立于固结于物体上的拉格朗日坐标系，这就使得前者更加适合流体问题的求解，而后者更适用于固体结构问题。

在有限单元法产生的历史进程中，传统的加权余量法与基于变分原理的里兹方法起到了关键性的支持作用。传统方法试图在整个求解域上直接假设近似函数并求解，这样的求解方法只适用于形态简单的几何形体。而有限单元法与它们的区别在于，将整个求解域划分为若干个单元，对各个单元域分别假设近似函数、分别求解后整合，这样就克服了几何形体的限制，适用于复杂形态的求解域。

2.2.2 有限元计算机分析流程[10]

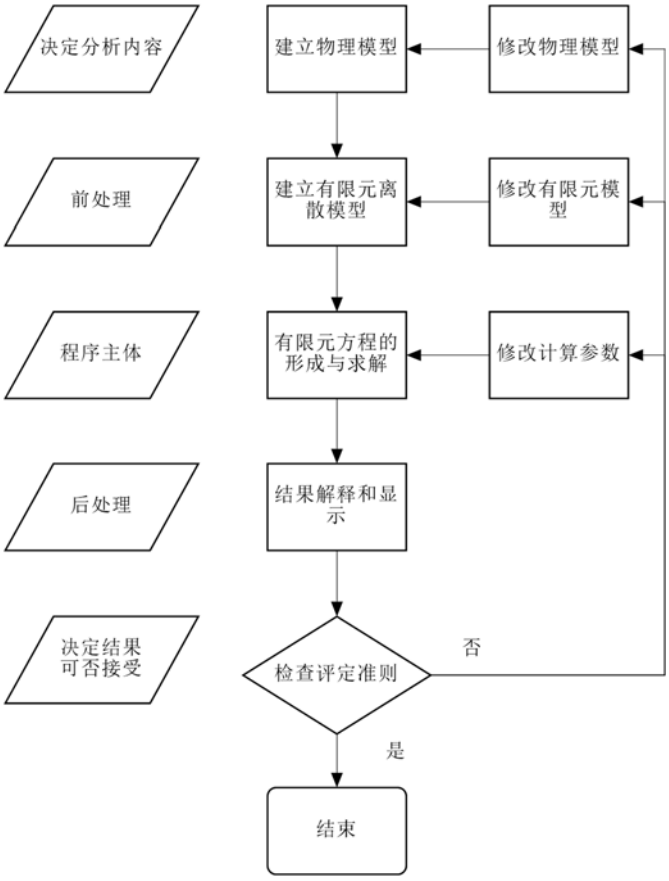


图 2.2 有限元计算及分析流程图

- 建立物理模型

将进行分析计算的目标桥梁结构的桥型、跨径、桥宽等物理特性输入到有限元分析计算机程序中，输入约束条件。

对于本论文中出现的物理模型，基于由浅入深、从简而繁的研究思路，选择以计算跨度为 50 米、主梁全长 51 米、桥宽 3.5 米、单主梁的预应力混凝土简支梁桥作为物理模型，预先输入特性。

- 建立有限元离散模型

将物理模型划分单元，得到离散的有限元模型。根据不同的求解域和不同的精度要求，选择单元的维度、形状、大小等方面。

- 有限元方程的建立和求解

选取多项式近似函数作为位移函数，建立位移与坐标之间的近似关系。

用矩阵表示即为：

$$u = \phi\beta \tag{3.1.1}$$

其中 u 为位移，ϕ 为位移模式，β 为广义坐标。

由于广义坐标 β 是待定的，将广义坐标 β 表示为位移 u 与位移模式 ϕ 的函数，再将所得的结果代入位移函数，得到位移函数的节点位移表示形式，其中节点位移的系数函数为形函数 N。用矩阵表示即为：

$$u = Na^e \tag{3.1.2}$$

其中 a^e 为单元节点位移列阵。

通过几何方程得到单元的应变 ε。用矩阵表示为

$$\varepsilon = Ba^e \tag{3.1.3}$$

其中 B 为应变矩阵，或称几何矩阵。

通过物理方程得到单元的应力 σ。用矩阵表示为

$$\sigma = Sa^e \tag{3.1.4}$$

其中 S 为应力矩阵。

- 结果解释和显示

  将上一步中的各个结点位移、各单元应力，通过图像加以显示。具体的显示方式有：等值线、带状云图、连续云图、数值显示等。

- 检查评定准则

  对计算的结果进行评判，如应力、应变是否超过阈值等。

3 荷载生成与效应分析

3.1 车辆荷载参数

3.1.1 参数需求

针对每一组车辆荷载，需要采集的参数包括：车辆编号、车速、车重、轴数、轴距、轴重、预计上桥时间、预计下桥时间、实际上桥时间、实际下桥时间。所有装置从上桥前若干距离就须设置，为达到提前预警的目的预留时间。

- 车辆编号：

主要功能为识别车辆。有两种采集记录方式：

一种是通过摄像头，自动识别车辆的牌照，理论上具有唯一性。然而在实际应用中，不能避免虚假牌照的情况，尤其是当研究视角将精细化的车辆荷载监控包容至小型车辆时。当遇到虚假牌照，存在使牌照信息相同而有两组荷载同时在桥上的可能性。

另一种为，当车辆被监测到时，记录该时刻的时间（精确至毫秒），通过对时间戳添加统一的备注后得到的字符串作为车辆的编号。这样操作可以保证车辆编号的绝对唯一性，不会因社会性的局外因素所干扰。

通过将两种记录方式同时应用，并叠加二者的字符串，可以得到信息最大化的、全面完整的车辆编号，在大数据时代，方便日后对有隐藏价值的车辆监控信息做其他方面更深入的研究。

- 车速：

通过设置测速仪，测得车辆在测速点的瞬时速度。在本次研究中，由于采用了经典理论的解析法进行计算，故仅考虑单一测点速度拾取，即速度不变的情况。在实际的项目展望中，希望实现通过有限元计算得到的精确预测，这样就对速度的准确性提出了要求。具体来说，需要在全桥布置若干个测速点，在一定密度上获取每一辆车的各位置、各时刻速度，并在车辆经过每一个测点时更新其在在该位置的速度，作为其到下一个测点前的平均速度，同时保留之前测点下与当前速度不同速度所引起的效应作为新的测点段的阶段性初始条件。如此，可以在速度层面达成对荷载的实时监控。

- 车重：

通过地磅，即汽车衡，测得通过的车辆荷载值，亦可换算为车重。为使得避免社会性问题（如作弊等人工干扰手段）对监控效果的影响，同时保证车牌捕获、通过时间、测点初段初始速度的信息对应性、有效性，拟采用“不停车快速超限检测系统”，获取移动车辆荷载在动态运输中的荷载。

由于移动中的荷载会对检测装置产生动力效应，使得直接获取的荷载信息偏大，无法达到精细化监控的目的，故需要对传感器荷载值进行修正。关于修正该荷载动态效应的捕获算法，并非本研究的重点，并且近年已有相应的不停车检测系统实际投入市场使用，本研究默认已获取准确的车辆荷载值。

- 轴距与轴数：

根据市场上已有的产品，基于红外线检测原理，通过其智能轮轴识别器，正确检测每一辆车的轴数[11]。

将车轴数信息中的时刻信息结合车速信息，可以得到车轴与车轴之间的距离，并可按短轴距、中轴距、长轴距三种规格分类。同时可以通过红外光设备对车辆进行轮廓扫描、车型识别，实现客货分离[11]。针对于本研究，需要尽量精准的轴距与轴数信息，即可使用轮廓扫描及车型识别信息对轴距与轴数进行修正，保证检测荷载信息的准确性。

- 轴重：

根据市场上已有的实际产品，可以使用轴载谱仪获得实时的分车道及各轴轴重信息[11]。

- 预计、实际上下桥时间

在车辆荷载上桥前若干距离l_0前，在统一位置集中设置的摄像头、初始测速仪获得v_0、汽车衡、轮轴识别仪、轴载谱仪，记录该时刻为监控起始时间t_0。计算t_0+l_0/v_0为预计上桥时间，计算$t_0+(l_0+l)/v_0$为预计下桥时间，并在每经过一个上桥前测点处获取t_n,v_n,l_n，进行预计上下桥时间的修正。通过在桥头与桥尾位置设置摄像头、测速仪，捕获实际的t_0,v_0,t_N,v_N作为实际上下桥时间。

3.1.2 规范与实际市场情况

高速公路、干线一级公路的设计速度为 100km/h，地质地形条件受限时采用 80km/h[12]，换算为国际单位制分别为 27.78m/s 与 22.22m/s。以该两速度分别作为本次研究荷载模拟生成的一快一慢两个模拟速度。

车辆荷载的设计值为 55 吨，即 550kN[13]。然而本研究须考虑实际复杂情况，包括超载车辆、特种车辆、多车辆的叠加作用，不能单纯以规范标准值作为模拟对象，故模拟设置轻车与重车两种标准。以实际投入市场的中国重汽-斯太尔 M5G 箱型货车作为轻型车辆标准，其整备质量为 10.20 吨，满载标准质量为 25.00 吨，即 250kN。以实际投入市场的中国重汽 HOWO 矿用载货车作为重型车辆标准，其整备质量 29.30 吨，满载质量 70.00 吨，即 700kN[14]。

3.2 荷载生成的一般考虑

模拟荷载的生成可以分为若干个层次。本研究着眼于成果的实现，采取由简入繁、由易

入难，由浅入深的层级进行研究。

- 首先，面对抽象的一维线形桥梁，生成单一匀速移动荷载集中力，这是车辆荷载的最简化形式，也是研究的起始点。
- 而后，将若干个不同速度的单一匀速移动荷载集中力依次在不同时刻、不同方向释放生成，得到多个匀速移动荷载集中力。然而在这里并没有涉入桥梁的横向刚度以及车道关系，继续使用抽象简化的一维线形进行讨论。由于本次研究受限于客观因素，最终成果展现至本层级，仅对更加深入的若干层级进行分析与讨论。
- 进一步地，设置轴距，将多个匀速移动的集中力按轴距距离以相同速度在同侧同时释放生成，得到匀速移动荷载集中力组，每一个力组代表着一辆实际的多轴车辆荷载。
- 再进一步地，将前两种思路整合，在不同方向、不同时刻生成若干不同速度的匀速移动车辆荷载力组。
- 接下来 ，由于在实际情况中，车辆并不会按匀速行驶，同时在本研究设想中，测速仪需要全桥间隔布置，于是需要将每个车速设置为在每个间隔节段内匀速而整体变速。变速的规则设置为在一定范围内生成随机数。就此，基本完成对一维线形桥梁模型下的真实车辆荷载生成模拟。
- 最后，将一维桥梁扩展至二维平面具有车道以及横向刚度的桥梁。需对车辆之间的位置关系、变道信息进行碰撞避让处理，防止在同一车道、同一位置产生车辆荷载的碰撞叠加。

3.3 效应分析

效应分析可分解为模态分析与振型分析两部分。本研究通过多种计算途径，对基于经典理论建立的动力学方程进行模态分析，其形式为二阶常系数非齐次线性微分方程，对该方程求解。其中包括参考李国豪《桥梁结构稳定与振动》[8]以及 Anil K．Chopra《结构动力学-理论及其在地震工程中的应用》[9]的推导思路所得到的解析解；使用 python 语言中科学计算模块 scipy.integrate 内置的 odeint 函数[15]，通过 Adams–Moulton 方法求解，得到精确度极高的数值解。出于对程序运算的考虑，最大程度精简代码，加快运算速度，最终采用数值求解方法对模态求解。同时，已编写公式推导与已知解方法的代码作为前期探索性、阶段性成果。

4 面向对象的 Python 程序开发

4.1 Python 语言

在研究伊始，确定使用 Python 语言作为主要开发语言。Python 语言主要具有以下特点与优势：

- 面向对象编程（Object Oriented Programming）。[16]

程序的编写目的在于解决问题，本研究所面对的是生成移动车辆荷载并计算、显示桥梁效应的问题，在解决此问题的过程中所涉及的子问题组成了一个问题域。面向对象的程序设计即强调以问题域中的事物作为核心来认识问题、思考问题，将这些事物抽象为对象，以对象作为程序的基本构成单位。这使得可以在程序设计中使用人类自然的逻辑思维方法（如抽象、分类、继承等）更有效地将问题表达。在 Python 中，每一个定义都会成为一个对象。Python 对象的类、模块可以拥有极为清晰的程序结构，通过简洁的语法层次面向、通过、针对对象实现程序运行。[17]

- 开源。

Python 语言的开源特性使得研究在编程过程中不产生费用，同时得到社区性而非商业性的专业支持。通过阅读源代码，可以最大程度理解并解决开发过程中所遇到的各类问题。

- 可移植性与可混合性。

Python 语言标准采用可移植的 ANSI C 标准，可以在包括 Linux, Unix, Windows, MacOS, VxWorks, Cray 及 IBM 超级计算机等绝大部分主流平台上无差别运行。同时，Python 可以与其他语言所编写的组件连接，混合使用，可以较为便利的添加功能。作为快速高效的原型开发工具，Python 可以最快实现开发需求，而后转移至 C 语言。

- 易读性。

Python 中使用大量的英文原意单词，是所有汇编语言中可读性最高的代码。这个特性使得 Python 语言具有了更强大的重复使用性与可维护性，而非一次性代码。这对于结构健康监测系统的维护、升级、拓展提供了更高的可能性。

基于以上特性，Python 语言已广泛应用于金融、生物、数学、城市规划、教育、艺术等诸多领域，其中在前沿科研上应用最为广泛。在土木工程行业，Python 已有一定应用，如“对循环荷载下的结构开裂发展模拟分析”(An object-oriented approach for modeling and simulation of crack growth in cyclically loaded structures)等。[18]

4.2 程序结构

从实现的功能场景作为出发点，本程序拟设置六个线程，进行并发运行，分别为

- 计算分析（Analyst）
- 输入监控（Monitor）
- 时间协同（Scheduler）
- 荷载管理（Manager）
- 图像显示（Animator）
- 安全预警（Alert）

其中图像显示线程布置在主线程内。以下通过 UML（Unified Modeling Language）形式的顺序图对多线程之间的运行关系进行阐述。[17]

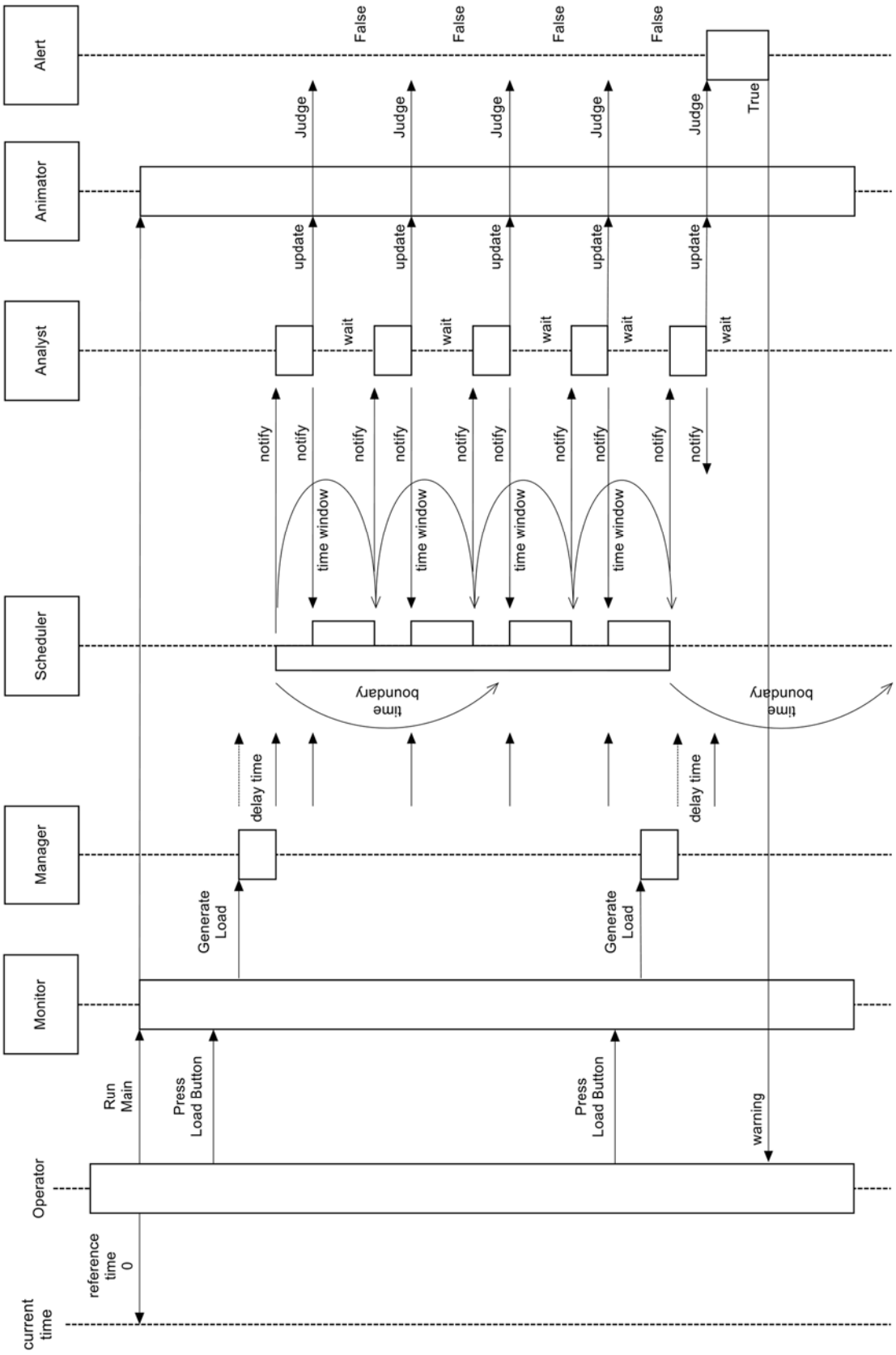


图 4.1 程序 UML 顺序图

- 输入监控线程的功能介绍

从程序起始开始监控键盘的输入，随时输出捕捉到的按键，输出为相应的荷载信息。目前设置 Q、A、P、L 四个键，分别作为左侧低速重车、左侧快速轻车、右侧低速重车、右侧快速轻车。

- 计算分析线程的功能介绍

将荷载、桥梁的基本信息输入计算分析线程，输出全桥每一时刻的效应值。计算机的计算速度可实现在若干毫秒完成全时间的计算，远远快于真实时间，每一次计算的使用时间是一个相对于真实时间的极小值。然而研究需要控制其进展，不能提前于真实时间之前将未输入荷载的时间时的效应计算完，否则会造成计算资源的浪费以及重复计算读写覆盖的混乱。这需要紧密的时间协同。

- 时间协同线程的功能介绍

为达成计算分析线程中控制时间的要求，充分控制计算时间与绘图时间。这里引入“时间界限”与“时间窗”的概念。

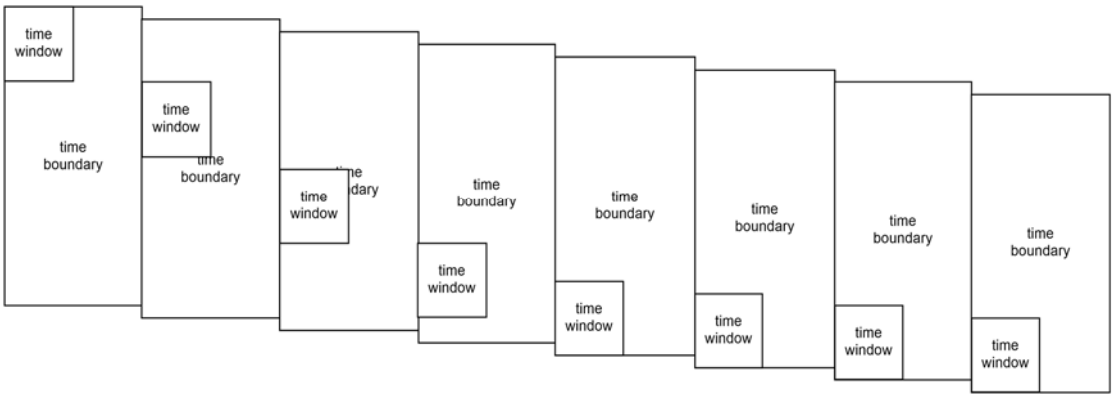


图 4.2 时间窗与时间界限

当车辆荷载进入引桥时，可通过其速度及引桥长度、主桥长度，判断其进入主桥的时刻、驶离主桥的时刻，即定义为时间界限的上限与下限。时间界限在时间轴上随真实时间不断平移，该段时间即针对该车辆荷载即将在桥上的行驶时间。在时间界限内划分若干个时间窗，每个时间窗是一次效应计算的最短时间单位。

- 荷载管理线程的功能介绍

当荷载信息被生成，判断荷载的预判时间是否在时间窗内。当荷载的预判时间进入时间窗内时，立刻将荷载导入计算分析线程，等待计算分析线程完成计算后，将效应结果输出。此时需要计算分析线程开始等待，同时激活时间协同线程，直到荷载的预判时间进入下一个时间窗。

- 图像显示线程的功能介绍

当每一个时间窗的求解结果被输出时，需要其被保存在一个内存空间中，而后图线显示线程从该空间中提取所需要效应信息。这里引入“效应池”的概念。

效应池是在图像显示线程下被定义的若干个效应值存储空间，本项目中缺省值为 6 个。当效应被计算分析线程输出后，将其次序放入效应池内。图像显示线程的读取速度是保持滞后于计算分析线程的，这样可以保证效应的读取与写入不发生冲突。

- 安全预警线程的功能介绍

随时循环检查，计算响应结果是否超过安全值。在响应结果未超过安全值时，保持监控。当响应结果超过安全值时，立刻触发相应的预警，并采取预警措施，如自动拦截、人为干预等。

5 研究结果

5.1 模型参数及生成

5.1.1 简支梁桥参数

本文采用 50 米主跨简支梁模型作为算例，检验本文方法。具体桥梁模型如下图所示：

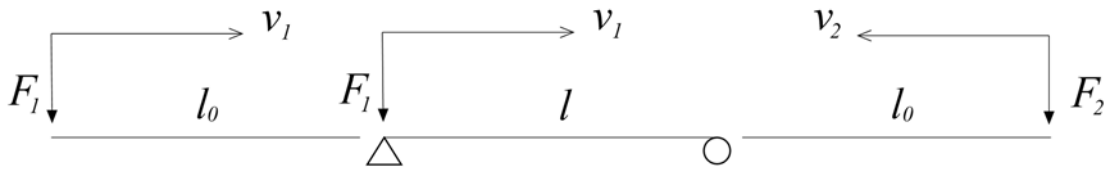


图 5.1 简支梁与车辆荷载的模型示意图

模型主要参数为：

参数	主跨长度 l	引桥长度 lo	截面刚度 EI	单位长度质量 m	阻尼系数
单位	m	m	kg m^3	kg/m	无量纲
参数值	50.0	50.0	4.53×10^{10}	5942.9	50000

表 5.1 简支梁桥模型主要参数

通过模态分析，可求出桥梁的前 3 阶模态频率，及相应的阻尼比。见下表：

模态阶数	频率 ω	阻尼比	振型函数
1	10.894763326874813	0.3860855936982675	$\sin\frac{\pi x}{l}$
2	43.579053307499251	0.096521398424566876	$\sin\frac{2\pi x}{l}$
3	98.052869941873297	0.042898399299807509	$\sin\frac{3\pi x}{l}$

表 5.2 前三阶固有频率及相应阻尼比

5.1.2 车辆荷载参数

车辆荷载参数的选取及考虑，由前述章节 3.1.2 给出。

5.1.3 车辆荷载模拟加载方式

方式一、采用按键方式生成，模拟实际上桥的车辆荷载

	Q	A	P	L	S	K	Z
大小 N	700000	250000	700000	250000	7000000	7000000	退出
速度 m/s	20	25	20	25	20	20	
方向	左至右	左至右	右至左	右至左	左至右	右至左	
形式	单一	单一	单一	单一	车队	车队	

表 5.3 车辆荷载模拟参数对应

方式二、采用车队(fleet)方式生成共振荷载

由章节 2.1.2 的分析表明，以匀速 v 移动的车辆集中力 F，对第 n 阶模态产生区间内的简谐荷载：

$$B_n(t)=\begin{cases}\dfrac{2F}{ml}\sin\dfrac{n\pi vt}{l}, 0\le t\le\dfrac{l}{v}\\ 0, \quad else\end{cases}$$

其理论上的共振条件为：

$$\frac{n\pi v}{l}=\omega_n=\left(\frac{n\pi}{l}\right)^2\sqrt{\frac{EI}{m}}$$

即当 $v=\dfrac{n\pi}{l}\sqrt{\dfrac{EI}{m}}$ 时，将激起第 n 阶模态共振。但实际上，单一车辆在桥梁行驶的时间有限，其荷载作用持时仅为：

$$T=\frac{l}{v}=\frac{n\pi}{\omega_n}=\frac{n}{2}T_n$$

这表明，荷载作用持时仅为第 n 阶模态周期的 n/2 倍。因此，单一车辆即使速度达到了共振的条件，由于持时有限，并不会产生共振。关于这一点，在文献[8]李国豪《桥梁结构稳定与振动》中已有述及。但是，若行驶的车辆荷载以持时间隔成队列出现，将会形成长时间作用的周期荷载，这时将会在一定时间后通过周期激励激起共振。

以本算例的数据参数代入，可以得出前三阶振型的共振车速分别为：

$v_1=173.395543729$

$v_2=2v_1$

$v_3=3v_1$

此速度折合 624.2km/h，约为民航客机的飞行时速现阶段的公路铁路桥梁中移动荷载较难达到此速度。但是，若考虑周期激励效应，使车队荷载的上桥间隔时间与固有振动周期一致，同样会得到由共振因素引起的效应响应。

5.2 程序代码

程序分为 5 个文件模块实现，详见附录。

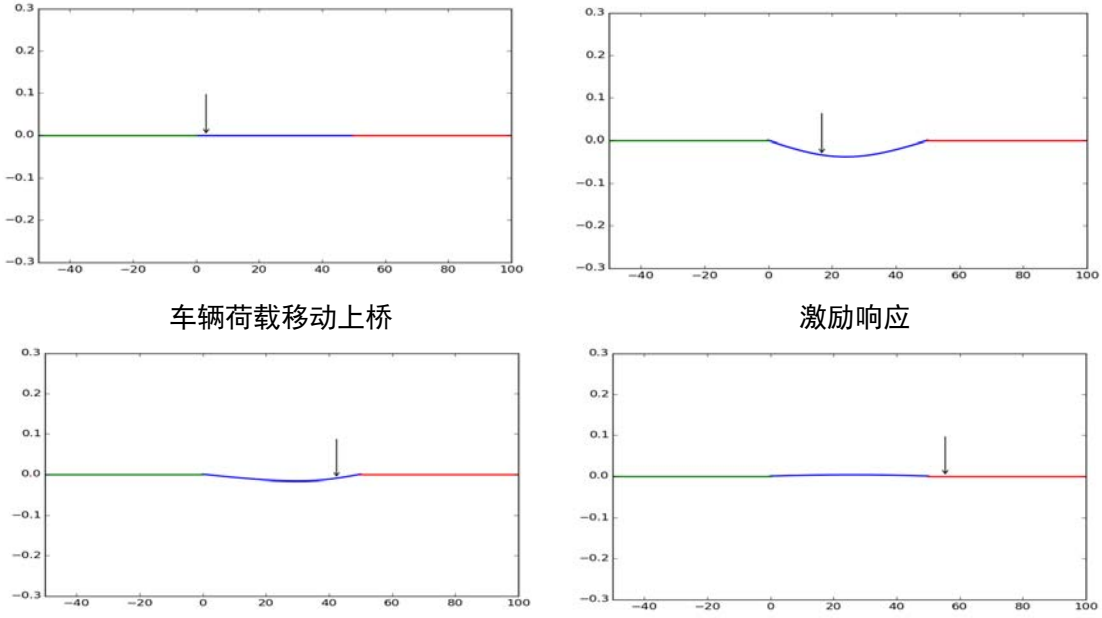
5.3 程序效果演示

5.3.1 Python console 输出内容示例

```
theBridge.omegaList =  [10.894763326874813, 43.579053307499251, 98.052869941873297]
theBridge.zetaList  =  [0.3860855936982675, 0.096521398424566876, 0.042898399299807509]
q
Analyst: wait... time_cur =  0.000000,  time_window = ( 0.000000,  0.000000), time_boundary =
( 0.000000,  0.000000)

Analyst: work... time_cur =  0.400000,  time_window = ( 1.900000,  2.400000), time_boundary =
( 0.400000,  2.400000)

idx_twin =  0,  idx_pool = 0,  time_window =( 1.900,  2.400)

for time_window(  1.900,  2.400), odeint using time = 0.000000 (sec)
                                    total  using time = 0.001000 (sec)

Analyst: wait... time_cur =  0.400000,  time_window = ( 1.900000,  2.400000), time_boundary =
( 0.400000,  2.400000)

Analyst: work... time_cur =  0.900000,  time_window = ( 2.400000,  2.900000), time_boundary =
( 0.900000,  2.900000)

idx_twin =  1,  idx_pool = 1,  time_window =( 2.400,  2.900)

for time_window(  2.400,  2.900), odeint using time = 0.000000 (sec)

                                    total  using time = 0.002000 (sec)

Analyst: wait... time_cur =  0.900000,  time_window = ( 2.400000,  2.900000), time_boundary =
( 0.900000,  2.900000)
z
Monitor: press key Z
```

5.3.2 动态效果展示

- 单一集中匀速荷载集中力作用：

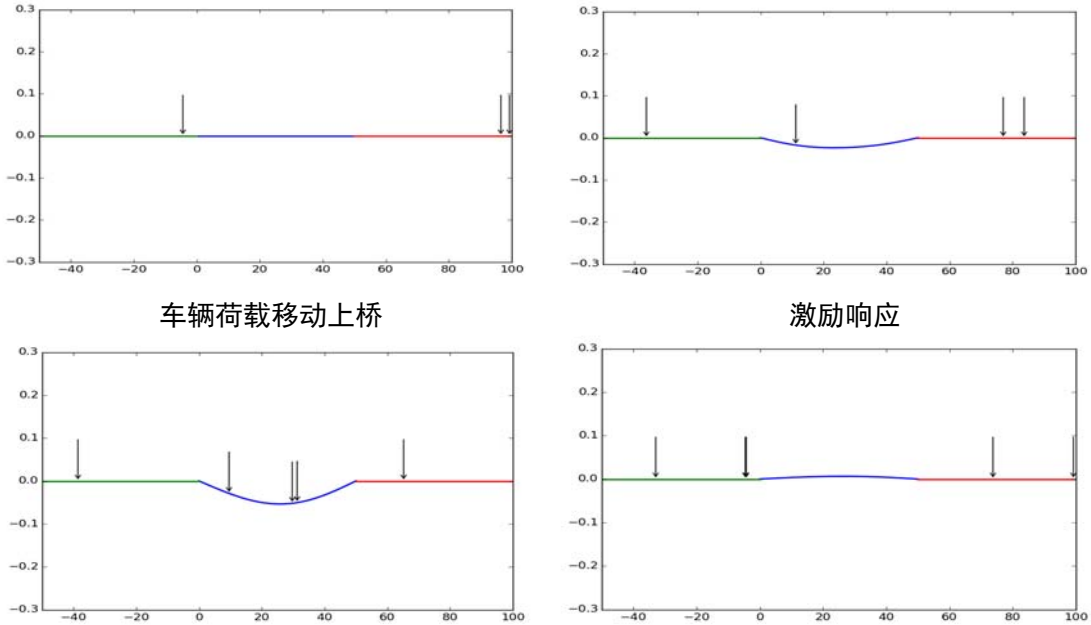


车辆荷载移动上桥　　激励响应

响应持续变化，直至驶离桥梁　　荷载离开后桥做衰减自由振动

图 5.2 单一集中匀速荷载集中力作用

- 若干随机匀速荷载集中力组合作用：



车辆荷载移动上桥　　激励响应

响应持续变化，直至驶离桥梁　　荷载离开后桥做衰减自由振动

图 5.2 多个随机集中匀速荷载集中力共同作用

5.3.3 各阶模态响应图

图中，红色为一阶振动模态响应，蓝色为二阶振动模态响应，黑色为三阶振动模态响应。

- 单一集中匀速荷载集中力作用：

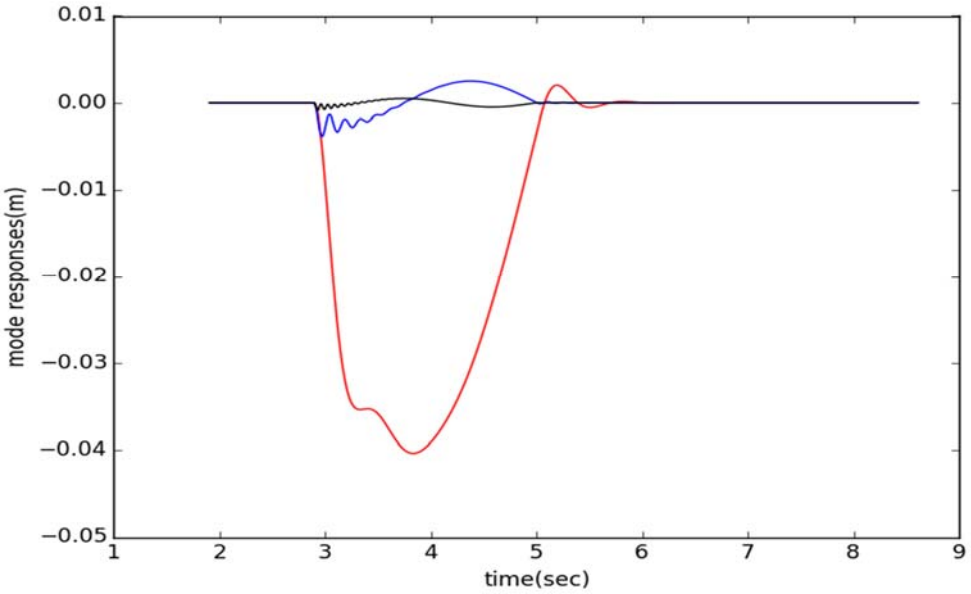


图 5.3 单一集中匀速荷载集中力所引起的模态响应

- 若干随机匀速荷载集中力组合作用：

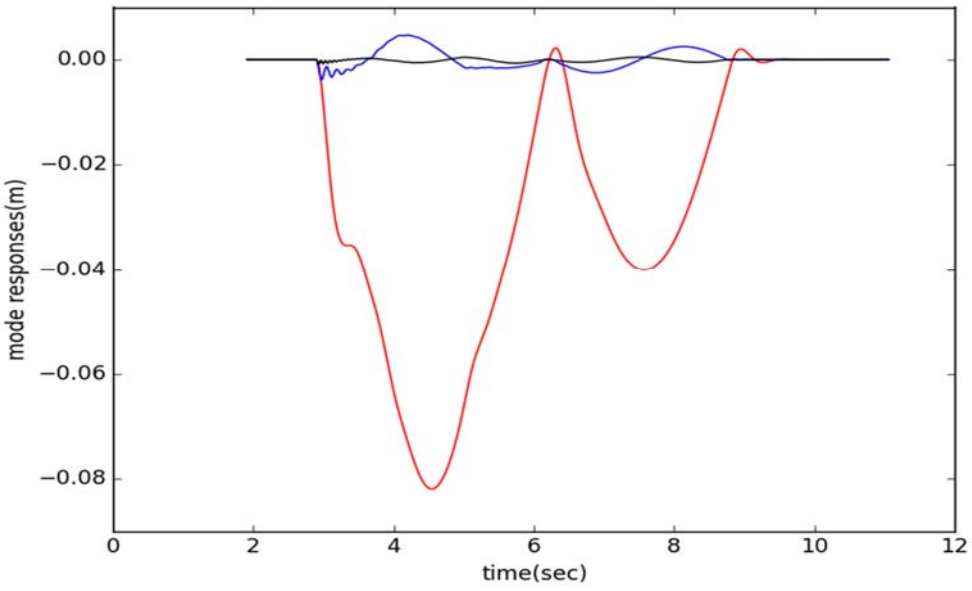


图 5.4 多个随机匀速荷载集中力组合作用所引起的模态响应

- 车队荷载作用

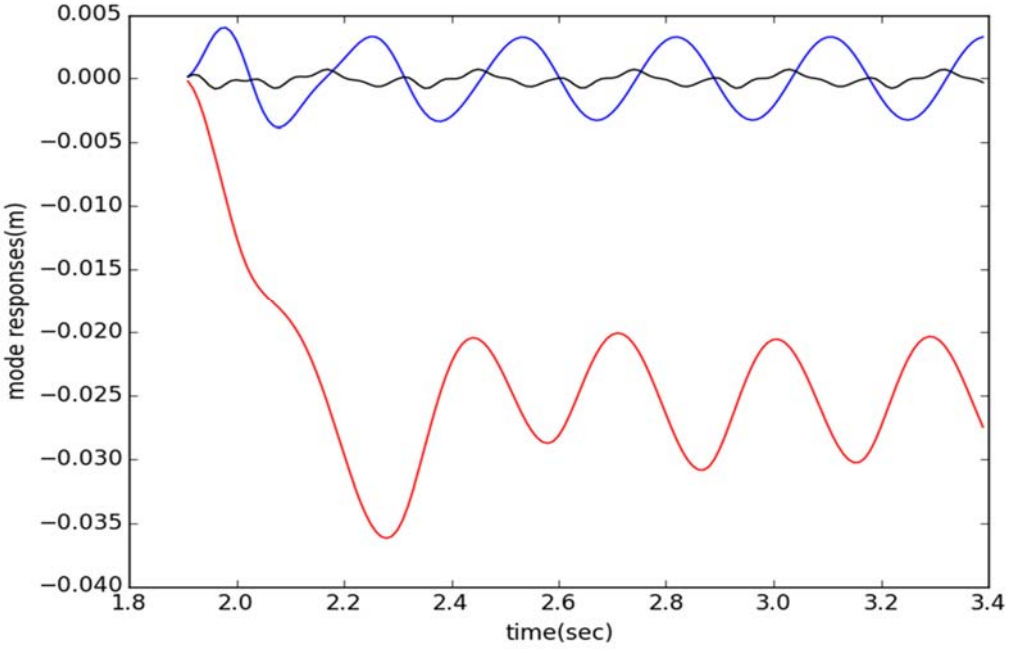


图 5.5 车队荷载作用下的模态响应

6 结论与展望

6.1 结论

本研究完成了桥梁移动车辆的实时监测生成及分析预警系统的概念阐述，并着重通过程序代码实现了匀速移动车辆荷载的实时生成与实时运算，实现了桥梁响应与移动车辆荷载的实时动态图形播放，初步证实了主动化桥梁结构健康监测系统的可行性，基本表述了其工程价值。

6.2 展望

千百年来，桥梁作为人类在地球上所构筑的人造物，是人类文明历史长河中最古老而恒久的一份子，桥梁是人类精神的象征。人类从未停止对桥梁可能性的探索，孜孜不倦地，从结构、材料、荷载、施工方法、设计手段等方面，人类创造着一项又一项动人心魄的桥梁奇观。

这是一个大数据的时代，这是一个物联网的时代。进无止境，勇往直前的人们，从未错过任何一个时代，也绝对不会错过当下的这个时代。程序的本质是逻辑演绎的形式化表达，记载的是人类对这个世界的数字化理解。而信息模型的建立，是为了帮助人们更好地认知真实世界中的事物。动态交通车辆监控信息与桥梁结构计算模型的结合，会使得桥梁在健康监控系统下融入物联网。在虚拟世界中构建如同真实的桥梁，驰骋着如同真实的车辆，模拟出如同真实的效应，这让一切更可知、更可控，让真实世界中的人们有所参考。进一步地，车流涌过，它们所留下的信息价值并不在于信息本身，更多的是其庞大的数据体量背后的更多可能性。

人们不会满足于仅仅跨过一条黄浦江，桥梁跨越长江、跨越山川、跨越海峡、跨越国界、跨越洲际，跨越了阻挡在人们前行途中的一切障碍。人们不会满足于繁复的满堂支架，桥梁有了挂篮、有了顶推、有了转体，有了更高效、更大胆、更富创造力的施工进程。现在，人们同样不会满足于目前对桥梁运营状态的被动监控，而是要主动出击，最大限度地获取、利用车辆的全方位信息。进一步地，甚至利用主动、精细化监控，自下而上影响由健康监控引导的精细化荷载新规范，引导实桥、真实车辆荷载主动模拟的全寿命桥梁设计新手段，并非天方夜谭。

展望未来，在时代的巨轮下，桥梁的发展始终如一，从未停步。

参考文献

[1]. HOUSNER G, BERGMAN L, CAUGHEY T, et al. Structural Control: Past, Present, and Future [J]. Journal of Engineering Mechanics, 1997, 123(9): 897-971.

[2]. BOLLER C, CHANG F, FUJINO Y. Structural Health Monitoring – An Introduction and Definitions [M]. United Kingdom: John Wiley & Sons, 2009.

[3]. 金小川, 周宗红, 金小安, 纪承子. 中国桥梁设计使用年限的研究 [J]. 公路与汽运, 2012, 03(2): 162-165.

[4]. 孙利民. 中国桥梁工程学术研究综述 2014: 桥梁健康监测 [J], 中国公路学报, 2014, 27(5): 54-59.

[5]. 张启伟. 大型桥梁健康监测概念与监测系统设计 [J]. 同济大学学报(自然科学版). 2001(01): 65-69.

[6]. 黄方林, 王学敏, 陈政清, 等. 大型桥梁健康监测研究进展 [J]. 中国铁道科学. 2005(02): 4-10.

[7]. 邬晓光, 徐祖恩. 大型桥梁健康监测动态及发展趋势 [J]. 长安大学学报(自然科学版). 2003(01): 39-42.

[8]. 李国豪. 桥梁结构稳定与振动 [M]. 北京: 中国铁道出版社 2002: 256-258, 292-296.

[9]. Anil K. Chopra. 结构动力学: 理论及其在地震工程中的应用: 第 2 版 [M]. 北京: 高等教育出版社, 2007：36-40, 48-57.

[10]. 王勖成. 有限单元法 [M]. 北京: 清华大学出版社, 2003: 14-129, 254-267.

[11]. 北京替帝西交通科技有限公司. 车辆分型的基本功能要求 [EB/OL]. (2014-04-05)[2015-05-25]. http://www.obchina.com/tdx/ask/2014/0405/158.html.

[12]. JTG D20-2006, 公路路线设计规范 [S]. 北京: 人民交通出版社, 2006.

[13]. JTG D60-2004, 公路桥涵设计通用规范 [S]. 北京: 人民交通出版社, 2004.

[14]. 中国重型汽车集团有限公司. 产品展厅: 载货汽车 [EB/OL]. (2015)[2015-06-02]. http://www.cnhtc.com.cn/View/ProductIndex.aspx.

[15]. Scipy developers. Scipy.integrate.odeint [EB/OL]. (2015-01-18)[2015-06-03]. http://docs.scipy.org/doc/scipy-0.15.1/reference/generated/scipy.integrate.odeint.html.

[16]. Robert Ian Mackie. Object-Oriented Programming of Distributed iterative equation solvers

[J]. Computer & Structures. 2008, 86: 511-519

[17]. 牛丽平, 郭新志, 宋强, 杨继萍 等. UML 面向对象设计与分析基础教程 [M]. 北京: 清华大学出版社, 2007: 2-119.

[18]. D. Cojocaru, A.M. Karlsson. An object-oriented approach for modeling and simulation of crack growth in cyclically loaded structures [J]. Advances in Engineering Software. 2008, 39(12).

7 致谢

由衷地感谢我的项目指导教师程纬老师对我的倾力指导与支持。针对本研究项目，程纬老师在指导我的过程中，投入了大量的时间与精力。在每周两次的座谈会面中，程纬老师填补我结构动力学与有限单元法方面的知识空缺，耐心回答我的每一个问题。在编程开发的后半程阶段，我遇到了始料未及的多线程并发协调问题，这对于现阶段的我的编程水平是力所不及的任务。幸而，程老师具有丰富的程序开发经验，在程纬老师的悉心示范、耐心指导下，本次研究才得以顺利实现。我想在此，对程纬老师致以我最真诚的谢意。

由衷地感谢我的母亲与父亲，感谢你们在大学期间对我的全力支持与信任。是我的父母长久以来、一如既往对我的物质与精神上无条件支持，使我可以心无旁骛地继续我的教育，将本论文完成。我也将一如既往地，尽全力为你们对我的付出负责，不辜负你们对我的期待。

由衷地感谢我身边的同学、朋友顾超、汤嘉熹、尹霄凡、倪逸凡等等。是与你们并肩共同奋斗的一个又一个日夜，构成了这篇论文的成文；与你们的相互鼓励，使我不再畏惧、逃避，让我勇于挑战自己。

由衷地感谢同济大学土木工程学院桥梁工程系，这里为我提供了强大而丰富的资源，提供了一个平坦宽广的平台，让我有幸结识包括程纬老师、朱乐东老师、肖汝诚老师、叶爱君老师、管仲国老师、周宗泽老师、薛二乐老师、方明山总工等等优秀的桥梁界先驱楷模，让我有充分的机会开拓的视野，充实自我。

再一次，用一句对我大学生活中所有人的感谢，完成我精彩纷呈的大学四年。谢谢！

8 附录 程序代码

8.1 文件一：vehicle.py

```
# coding: utf-8

import bridge
import numpy as np

# v_slow              = 200.0/9.0    # m/s
# v_fast              = 250.0/9.0    # m/s #
v_slow = 20    # 取简化值
v_fast = 25    # 取简化值

t_delay_slow = bridge.s_span / v_slow
t_delay_fast = bridge.s_span / v_fast

f_normal = 550000    # N # =>55t 规范
f_small = 250000    # N #满载质量 25t 轻车(small)
f_huge = 700000    # N #满载质量 70t 重车(huge)


class MovingConcentrateForce:
    def __init__(self, time_stamp, direction, vehicle_type, bridge, i=0):
        if vehicle_type == "sequence":
            self.name = "SQ_" + "%02i" % (i)
            self.force = f_huge
            omg = bridge.omegaList[0]
            l = bridge.m_span
            pi = np.pi
            velo_res = omg * l / pi    # 尝试激起一阶模态的共振

            self.velocity = velo_res
            self.delay = bridge.s_span / self.velocity
        elif vehicle_type == "sequence2":
            self.name = "SQ2_" + "%02i" % (i)
            self.force = f_huge
            omg = bridge.omegaList[1]
            l = bridge.m_span
            pi = np.pi
            velo_res = 2 * omg * l / pi    # 尝试激起二阶模态的共振
```

```
            self.velocity = velo_res
            self.delay = bridge.s_span / self.velocity
        elif vehicle_type == "huge_slow":
            self.name = "HS_" + "%010.3f" % (time_stamp)
            self.force = f_huge
            self.velocity = v_slow
            self.delay = t_delay_slow
        else:    # elif vehicle_type == "small_fast":
            self.name = "SF_" + "%010.3f" % (time_stamp)
            self.force = f_small
            self.velocity = v_fast
            self.delay = t_delay_fast

        self.direction = direction

        self.bridge = bridge
        self.m_span = bridge.m_span

        self.duration = self.m_span / self.velocity

        self.time_stamp = time_stamp + i * self.duration

    def Info(self):
        return (self.name, self.time_stamp, self.delay, self.duration, self.force, self.velocity,
self.direction,
                self.bridge.bridge_id)

    def OnBridgeCheck(self, time_line):    # 检查是否在桥上
        if time_line - self.time_stamp - self.delay > self.duration:    # 已离开
            return False
        if time_line - self.time_stamp - self.delay < 0:    # 未上桥
            return False
        return True    # 正在桥上

    def LeaveBridgeCheck(self, time_line):    # 检查是否离开桥
        if time_line - self.time_stamp - self.delay > self.duration:    # 已离开
            return True
        return False    # 未离开桥

    def NotReachBridgeCheck(self, time_line):    # 检查是否到达桥
        if time_line - self.time_stamp - self.delay < 0:    # 未到达
```

```
            return True
        return False    # 已到达

    def XPosition(self, time):
        x = self.velocity * (time - self.time_stamp - self.delay)
        if self.direction == "from_left":
            return x
        else:
            return self.m_span - x
```

8.2 文件二：bridge.py

```
# coding: utf-8

import numpy as np

m_span = 50.0    # m 主跨
s_span = 50.0    # m 边跨
E = 36500000000    # Pa # C65 混凝土
I = 1.24    # m^4
# T 形，上翼缘宽度 3.5m，梁高 3.3m，翼缘厚 0.2m，腹板厚 0.25m
m = 148600.0 / 25    # kg/m # (3.5*0.2+2.1*0.2)*50*26.0/9.8 # 预应力混凝土重力密度
26kN/m^3
damp_c = 50000


class Bridge:
    def __init__(self, bridge_id="", m_span=m_span, s_span=s_span, EI=E * I, m=m,
damp_c=damp_c):
        self.bridge_id = bridge_id
        self.m_span = m_span    # 中跨
        self.s_span = s_span    # 左右边跨
        self.EI = EI
        self.m = m
        self.damp_c = damp_c
        self.num_mode = 0

        self.omegaList = []    # 模态固有频率列表
        self.zetaList = []    # 模态阻尼比列表
        self.modeShapeList = []    # 模态向量列表（每个元素也是列表，维数与坐标维数
```

一致）

```
        self.x = None

    def SetSegmentNumber(self, num_segment=128):
        self.x = np.linspace(0, self.m_span, num_segment)

    def ModeAnalyze(self, num_mode=3):
        self.num_mode = num_mode
        for i in range(1, num_mode + 1):
            omega = (i * np.pi / self.m_span) * (i * np.pi / self.m_span) * np.sqrt(self.EI /
self.m)
            self.omegaList.append(omega)

            zeta = self.damp_c / self.m / 2.0 / omega
            self.zetaList.append(zeta)

            y = np.sin(i * np.pi * self.x / self.m_span)
            self.modeShapeList.append(y)
```

8.3 文件三：bvs.py

```
# coding: utf-8

import copy

import win32api
import pythoncom
import pyHook

import threading
import time

import numpy as np
from scipy.integrate import odeint
from matplotlib import pyplot as plt
from matplotlib import animation

import vehicle
import bridge
```

```
start_flag = False
finish_flag = False

sec_grade = 6  # 微秒

theBridge = bridge.Bridge("A_bridge")  # 运行桥梁
vehicle_dict = dict()  # 车辆字典

time_ref = 0  # 参考时间（初值零）

delay_min = vehicle.t_delay_fast  # 最小时间延迟
time_boundary = (0, 0)  # 时间边界（初值）

time_length = delay_min / 4.0  # 时间窗口长度（初值）
time_window = (0, 0)  # 时间窗口（初值）
num_seg_per_time_window = 50  # 每时窗分段数

A_pool = list()  # 模态坐标数据池
t_pool = list()  # 时间数据池
num_pool = 6  # 预留 num_pool 个数据池，供循环存放 num_pool 个时间窗口的模态响应数据
idx_pool = 0  # 当前数据池
idx_twin = 0  # 当前时间窗口序号

echo_flag_monitor = False
echo_flag_manager = False
echo_flag_schedul = False
echo_flag_analyst = False

file_monitor = None
file_manager = None
file_analyst = None
file_schedul = None
file_animate = None


def StrTimeInfo(caller=""):
    global idx_twin, idx_pool, time_ref, delay_min, time_boundary, time_length, time_window,
num_seg_per_time_window
    global sec_grade
    if caller == "Manager":
```

```
        if sec_grade == 3:    # 毫秒级
            fmt = "    time_window = (%6.3f, %6.3f)\n"
        else:    # 微秒级
            fmt = " time_window = (%9.6f, %9.6f)\n"
        return fmt % (time_window[0], time_window[1])

    if sec_grade == 3:    # 毫秒级
        fmt = "    time_cur = %6.3f,     time_window = (%6.3f, %6.3f),    time_boundary =
(%6.3f, %6.3f)\n"
    else:    # 微秒级
        fmt = " time_cur = %9.6f,    time_window = (%9.6f, %9.6f), time_boundary =
(%9.6f, %9.6f)\n"

    return fmt % (time_boundary[0], time_window[0], time_window[1], time_boundary[0],
time_boundary[1])


class Monitor(threading.Thread):
    def __init__(self, cond, lock, name='Monitor'):
        threading.Thread.__init__(self)
        self.cond = cond
        self.name = name
        self.lock = lock

    def run(self):
        global file_monitor, echo_flag_monitor
        global start_flag, finish_flag

        file_monitor = open("output_monitor.txt", "w")
        if echo_flag_monitor:
            print "Monitor: key board hook OK."

        hm = pyHook.HookManager()    # 创建钩子管理对象
        hm.KeyDown = self.OnKeyBoardEvent    # 监听键盘事件

        hm.HookKeyboard()    # 设置键盘钩子
        pythoncom.PumpMessages()    # 进入循环

        if echo_flag_monitor:
            print "Monitor: exit."
        file_monitor.close()
```

```
        finish_flag = True

    def OnKeyBoardEvent(self, event):
        global vehicle_dict, theBridge
        global idx_twin, idx_pool, time_ref, delay_min, time_boundary, time_length,
time_window, num_seg_per_time_window
        global file_monitor, echo_flag_monitor
        global start_flag, finish_flag

        if start_flag == False:
            start_flag = True
            time_ref = round(time.time(), sec_grade)

        time_stamp = round(time.time() - time_ref, sec_grade)

        if event.KeyID == 81:    # Q
            vehicle_force = vehicle.MovingConcentrateForce(time_stamp, "from_left",
"huge_slow", theBridge)
            vehicle_id = "HS_" + "%010.3f" % (time_stamp)
        elif event.KeyID == 65:    # A
            vehicle_force = vehicle.MovingConcentrateForce(time_stamp, "from_left",
"small_fast", theBridge)
            vehicle_id = "SF_" + "%010.3f" % (time_stamp)
        elif event.KeyID == 80:    # P
            vehicle_force = vehicle.MovingConcentrateForce(time_stamp, "from_right",
"huge_slow", theBridge)
            vehicle_id = "HS_" + "%010.3f" % (time_stamp)
        elif event.KeyID == 76:    # L
            vehicle_force = vehicle.MovingConcentrateForce(time_stamp, "from_right",
"small_fast", theBridge)
            vehicle_id = "SF_" + "%010.3f" % (time_stamp)
        elif event.KeyID == 75:    # K
            self.lock.acquire()    # 加锁,避免冲突
            for i in range(200):
                vehicle_force = vehicle.MovingConcentrateForce(time_stamp, "from_left",
"sequence2", theBridge, i)
                info = vehicle_force.Info()
                vehicle_id = info[0]
                vehicle_dict[vehicle_id] = vehicle_force    # 修改 vehicle_dict
                file_monitor.write(str(vehicle_force.Info()) + "\n")
```

```
                    if echo_flag_monitor:
                        print ", remain", vehicle_dict.keys()
                        print
                self.lock.release()   # 释放锁
            elif event.KeyID == 83:   # S
                self.lock.acquire()   # 加锁,避免冲突
                for i in range(200):
                    vehicle_force = vehicle.MovingConcentrateForce(time_stamp, "from_left",
"sequence", theBridge, i)
                    info = vehicle_force.Info()
                    vehicle_id = info[0]   # "SQ_" + "%02i" % (i)
                    vehicle_dict[vehicle_id] = vehicle_force   # 修改 vehicle_dict
                    file_monitor.write(str(vehicle_force.Info()) + "\n")
                    if echo_flag_monitor:
                        print ", remain", vehicle_dict.keys()
                        print
                self.lock.release()   # 释放锁
            elif event.KeyID == 90:   # Z
                print "Monitor: press key Z"
                win32api.PostQuitMessage()
                return True
            else:
                return True

            file_monitor.write(str(vehicle_force.Info()) + "\n")

            if echo_flag_monitor:
                print
                print "Monitor: time_stamp =", time_stamp,
                print ", new vehicle =", vehicle_force.Info(),

            self.lock.acquire()   # 加锁,避免冲突
            vehicle_dict[vehicle_id] = vehicle_force   # 修改 vehicle_dict
            if echo_flag_monitor:
                print ", remain", vehicle_dict.keys()
                print
            self.lock.release()   # 释放锁

            return True
```

```
class Manager(threading.Thread):
    def __init__(self, cond, lock, name='Manager'):
        threading.Thread.__init__(self)
        self.cond = cond
        self.name = name
        self.lock = lock

    def SaveVehicleDict(self):
        for v_id in vehicle_dict.keys():   # 对荷载循环
            vehicle_force = vehicle_dict[v_id]
            fmt = "name =%s, stamp =%9.3f, delay =%9.3f, duration =%9.3f, force =%9.3f,
velo =%9.3f, direction =%s, bridge_id =%s\n"
            file_manager.write(fmt % vehicle_force.Info())

    def run(self):
        global vehicle_dict, theBridge
        global idx_twin, idx_pool, time_ref, delay_min, time_boundary, time_length,
time_window, num_seg_per_time_window
        global echo_flag_manager, file_manager
        global sec_grade
        global start_flag, finish_flag

        file_manager = open("output_manager.txt", "w")

        while True:
            if start_flag == True: break   # 开始

        if echo_flag_manager:
            print "Manager: ready..., ",
            print StrTimeInfo()
            print

        while True:
            v_id_set = set()   # 空集合

            self.lock.acquire()   # 加锁
            for v_id in vehicle_dict.keys():   # 对荷载循环
                vehicle_force = vehicle_dict[v_id]
                if vehicle_force.LeaveBridgeCheck(time_window[0]):   # 检查是否离开桥
梁
                    v_id_set.add(v_id)   # 放入已离桥车辆集合
```

```
                for v_id in v_id_set:
                    TT = vehicle_dict[v_id].duration + vehicle_dict[v_id].delay
                    if echo_flag_manager:
                        print "Manager: dele..., ", v_id, ", TT = ", TT,
                        print StrTimeInfo("Manager")
                        print
                        #                       del vehicle_dict[v_id]
# 从字典中删除移动车辆荷载
                self.lock.release()   # 释放锁

                if finish_flag == True:
                    self.SaveVehicleDict()
                    break   # 结束

            file_manager.close()


class Schedul(threading.Thread):
    def __init__(self, cond, lock, name='Schedul'):
        threading.Thread.__init__(self)
        self.cond = cond
        self.name = name
        self.lock = lock

    def run(self):
        global idx_twin, idx_pool, time_ref, delay_min, time_boundary, time_length,
time_window, num_seg_per_time_window
        global echo_flag_schedul
        global file_schedul, start_flag

        file_schedul = open("output_schedul.txt", "w")

        while True:
            if start_flag == True:    break   # 开始

        delta_t = 0.1   # 积分求解的时间提前量，便于检验初始条件是否正确
        while True:
            time_cur = time.time() - time_ref
            if time_cur >= time_length - delta_t:     break   # 时间前进一个时间窗
```

```
        self.cond.acquire()

        time_front = time_length - delta_t + delay_min    # 时间前沿
        time_window = (time_front - time_length, time_front)    # 首时间窗
        time_boundary = (time_front - delay_min, time_front)    # 首时间边界
        idx_twin = 0    # 时间窗口序号
        idx_pool = idx_twin % num_pool

        while True:
            self.cond.notify()    # 发出通知
            if echo_flag_schedul:
                print "Schedul: wait...",
                print StrTimeInfo()
                print

            new_time_lower = time_window[1]    # 新时间窗下限
            new_time_upper = new_time_lower + time_length    # 新时间窗上限

            self.cond.wait()    # 等待通知
            file_schedul.write(StrTimeInfo())
            if echo_flag_schedul:
                print "Schedul: work...",
                print StrTimeInfo()

            while True:
                time_cur = round(time.time() - time_ref, sec_grade)    # 当前时间
                time_front = time_cur + delay_min    # 前沿时间
                time_boundary = (time_cur, time_front)

                if new_time_upper <= time_front:    # 时间窗口(time_lower, time_upper)必
须介于(time_cur, time_front)之间，将时间窗口尽可能前移
                    time_window = (new_time_lower, new_time_upper)
                    idx_twin += 1
                    idx_pool = idx_twin % num_pool
                    break    # 跳出内层 while

            if finish_flag == True:
                break

        file_schedul.close()
```

```
            self.cond.release()


class Analyst(threading.Thread):
    def __init__(self, cond, lock, name='Analyst'):
            threading.Thread.__init__(self)
            self.cond = cond
            self.lock = lock
            self.name = name
            self.v_dict = dict()

    def run(self):
            global vehicle_dict, theBridge
            global idx_twin, idx_pool, time_ref, delay_min, time_boundary, time_length,
time_window, num_seg_per_time_window
            global echo_flag_analyst, file_analyst
            global A_pool, t_pool, num_pool, idx_pool
            global start_flag, finish_flag

            file_analyst = open("output_analyst.txt", "w")   # 打开工作文档

            while True:
                if start_flag == True:   break   # 开始

            self.cond.acquire()   # condition 启动

            num_mode = theBridge.num_mode
            t_num = num_seg_per_time_window

            y0 = [0.0] * (2 * num_mode)   # 分析开始时，零初始条件

            file_analyst.write(" time              A1                          A2
A3\n")
            while True:
                if echo_flag_analyst: print "Analyst: wait..." + StrTimeInfo()

                self.cond.wait()   # 等待 schedul 的通知
                if echo_flag_analyst: print "Analyst: work..." + StrTimeInfo()

                analyst_t0 = time.time()
```

```
            self.lock.acquire()  # 加锁,避免分析时与 manager 冲突
            self.v_dict = copy.deepcopy(vehicle_dict)  # 深拷贝，避免引用式共享
            self.lock.release()  # 释放锁

            (t0, t1) = time_window  # 取时间窗的上下限
            t_array = np.linspace(t0, t1, t_num + 1)  # 离散时间数组

            analyst_t1 = time.time()

            y_array = odeint(self.dydt, y0, t_array)  # 调用 odeint，求解各模态响应的时程
曲线

            analyst_t2 = time.time()

            # 保存到数据池
            if echo_flag_analyst: print  "idx_twin =%6i,  idx_pool =%2i,  time_window
=(%6.3f, %6.3f)\n" % (
                idx_twin, idx_pool, t0, t1)
            pos = idx_pool * t_num
            for i in range(t_num):
                t_pool[pos + i] = t_array[i + 1]
                A_pool[pos + i][:] = y_array[i + 1][0:(2 * num_mode):2]  # 仅取模态坐
标，不取其导数

            for i in range(t_num):
                #                 fmt = "      odeint:    idx_pool = %2i,      idx = %4i,
time = %6.3f,    A1 =%13.6e,    A2 = %13.6e,    A3 = %13.6e\n"
                #                 file_analyst.write( fmt % (idx_pool, pos+i, t_array[i+1],
y_array[i+1][0], y_array[i+1][2], y_array[i+1][4]))
                fmt = "%6.3f,          %13.6e,          %13.6e,          %13.6e\n"
                file_analyst.write(fmt % (t_array[i + 1], y_array[i + 1][0], y_array[i + 1][2],
y_array[i + 1][4]))

            # fmt = "      t_pool A_pool:                                    time = %6.3f,
A1 =%13.6e,    A2 = %13.6e,    A3 = %13.6e\n\n"
            #                 file_analyst.write( fmt % (t_pool[pos+i], A_pool[pos+i][0],
A_pool[pos+i][1], A_pool[pos+i][2]))

            idx_pool = (idx_pool + 1) % num_pool  # 指向下一个数据池，往复循环
            y0 = y_array[-1]  # 设置下一个时间窗的初始条件
```

```
                self.cond.notify()  # 通知 schedul
                analyst_t3 = time.time()

                using_time = round(analyst_t2 - analyst_t1, 6)
                if echo_flag_analyst: print  "for time_window(%9.3f, %9.3f), odeint using time
=%9.6f (sec)\n" % (
                    t0, t1, using_time)
                using_time = round(analyst_t3 - analyst_t0, 6)
                if echo_flag_analyst: print  "                                                  total
using time =%9.6f (sec)\n" % (
                    using_time)
                if finish_flag == True:
                    print "analyst finish."
                    break

            file_analyst.close()

            self.cond.release()  # condition 释放

        def dydt(self, y, t):  # 未知函数一阶导数 dydt
            global theBridge

            omega = theBridge.omegaList
            zeta = theBridge.zetaList
            num_mode = theBridge.num_mode

            y_p = [0.0] * (num_mode * 2)

            m = theBridge.m
            l = theBridge.m_span  # 主跨长度
            pi_l = np.pi / l
            ml = m * l

            for i in range(num_mode):  # 对模态循环
                omg = omega[i]  # 取当前圆频率
                zt = zeta[i]  # 取当前阻尼比
                zt2 = zt + zt

                i2 = i + i
                y_p[i2] = y[i2 + 1]
                y_p[i2 + 1] = (-zt2 * y[i2 + 1] - omg * y[i2]) * omg
```

```
        for v_id in self.v_dict.keys():   # 对车辆字典循环
            mv = self.v_dict[v_id]   # 当前移动车辆

            f = mv.force   # 荷载
            v = mv.velocity   # 速度
            t0 = mv.time_stamp + mv.delay   # 上桥时刻

            f2_ml = (f + f) / ml   # 2*f/m/l
            T = mv.duration
            piv_l = pi_l * v

            for i in range(num_mode):   # 对模态循环
                i2p1 = i + i + 1
                if mv.OnBridgeCheck(t):   # 当前是否在桥上
                    if mv.direction == "from_left":   # 方向：从左向右
                        y_p[i2p1] = y_p[i2p1] + f2_ml * np.sin((i + 1.0) * piv_l * (t - t0))
                    else:   # 方向：从右向左
                        y_p[i2p1] = y_p[i2p1] + f2_ml * np.sin((i + 1.0) * piv_l * (T - (t -
t0)))

        return y_p


fig = plt.figure()
ax = plt.axes(xlim=(-theBridge.s_span, theBridge.m_span + theBridge.s_span), ylim=(-0.3, 0.3))

line_m_brg, = ax.plot([], [], lw=2)   # 振动桥梁
line_l_brg, = ax.plot([-theBridge.s_span, 0], [0, 0], lw=2)
line_r_brg, = ax.plot([theBridge.m_span, theBridge.m_span + theBridge.s_span], [0, 0], lw=2)

arr_vehicle = [ax.annotate("", (0, 0), (0, 0), arrowprops=dict(arrowstyle="->")) for i in range(20)]
# 移动车辆


def InitAnimate():
    return line_m_brg, arr_vehicle


# play_animate function.    This is called sequentially
def UpdateAnimate(idx_f, num_mode, num_x, x, modeShapes, lock):
```

```
    global vehicle_dict, theBridge
    global idx_twin, idx_pool, time_ref, delay_min, time_boundary, time_length, time_window,
num_seg_per_time_window
    global A_pool, t_pool, num_pool, idx_pool, idx_twin
    global file_animate, echo_flag_animate
    global start_flag, finish_flag

    idx_t = (3 * idx_f) % (num_pool * num_seg_per_time_window)

    y = [0.0] * num_x

    t = t_pool[idx_t]
    A = A_pool[idx_t][:]

    file_animate.write("idx_f = %4i, idx_t =%4i,   time =%6.3f,     A1 =%13.6e,     A2
=%13.6e,     A3 =%13.6e\n" % (
        idx_f, idx_t, t, A[0], A[1], A[2]))

    for j in range(num_mode):
        y = y + A[j] * modeShapes[j]    # y = A1*sin(1*pi*x/l) + A2*sin(2*pi*x/l) +
A3*sin(3*pi*x/l) + ...

    line_m_brg.set_data(x, y)

    lock.acquire()   # 加锁,避免分析时与 manager 冲突
    v_dict = copy.deepcopy(vehicle_dict)   # 深拷贝
    lock.release()   # 释放锁

    m_span = theBridge.m_span
    #       arr_vehicle = list()

    i_mv = 0
    for v_id in v_dict.keys():
        mv = v_dict[v_id]
        x_mv = mv.XPosition(t)

        xlim = ax.get_xlim()
        x_l = xlim[0]
        x_r = xlim[1]

        if x_l <= x_mv and x_mv <= x_r:   # 在左引桥 + 主桥 + 右引桥上
```

```
                if mv.OnBridgeCheck(t):    # 在主桥上
                    itmp = np.floor(x_mv / (m_span / (num_x - 1.0)))    # 竖向位移 取 x_mv
坐标处的 挠度函数 插值
                    (x0, y0) = (x[itmp], y[itmp])
                    (x1, y1) = (x[itmp + 1], y[itmp + 1])
                    y_mv = ((x_mv - x0) * y1 - (x_mv - x1) * y0) / (x1 - x0)    # 插值
                else:
                    y_mv = 0    # 竖向位移取零

                # mv_arr = ax.annotate("", (0, 0),(0, 0), arrowprops=dict(arrowstyle="->"))    # 箭
头表示移动车辆
                #                 mv_arr.xy       = (x_mv, y_mv)
# arrow head
                #                 mv_arr.xyann = (x_mv, y_mv + 0.1)
# arrow tail

                #                 arr_vehicle.append(mv_arr)

                arr_vehicle[i_mv].xy = (x_mv, y_mv)    # arrow head
                arr_vehicle[i_mv].xyann = (x_mv, y_mv + 0.1)    # arrow tail
            else:
                arr_vehicle[i_mv].xy = (101, 0)    # arrow head
                arr_vehicle[i_mv].xyann = (101, 0.1)    # arrow tail
            i_mv += 1

    if finish_flag == True:    file_animate.close()

    return line_m_brg, arr_vehicle


# call the animator.    blit=True means only re-draw the parts that have changed.
def ShowAnimate(lock):
    global idx_twin, idx_pool, time_ref, delay_min, time_boundary, time_length, time_window,
num_seg_per_time_window
    global file_animate, echo_flag_animate
    global start_flag, finish_flag
    global idx_pool, idx_twin

    file_animate = open("output_animate.txt", "w")
    file_animate.write("output_animate.txt opened.\n")
```

```
    while True:
        if start_flag == True:    break

    time.sleep(0.5)

    num_mode = theBridge.num_mode
    x = theBridge.x
    num_x = len(x)
    modeShapes = theBridge.modeShapeList

    anim = animation.FuncAnimation(fig, UpdateAnimate, init_func=InitAnimate,
                                   fargs=(num_mode, num_x, x, modeShapes, lock,),
interval=3, blit=False)

    plt.show()


def MasterController():
    global vehicle_dict, theBridge
    global idx_twin, idx_pool, time_ref, delay_min, time_boundary, time_length, time_window,
num_seg_per_time_window
    global echo_flag_monitor, echo_flag_manager, echo_flag_schedul, echo_flag_analyst
    global t_pool, A_pool, num_pool, idx_pool, idx_twin



    #       echo_flag_monitor = True
    #       echo_flag_manager = True
    #       echo_flag_schedul = True
    echo_flag_analyst = True

    theBridge = bridge.Bridge("A_bridge")   # 生成桥梁
    theBridge.SetSegmentNumber(100)   # 缺省 128 段
    theBridge.ModeAnalyze(3)   # 缺省 3 阶模态分析

    num_mode = theBridge.num_mode   # 取模态数

    #           omg = theBridge.omegaList[-1]                    # 最高阶模态频率
    #           period = 2.0*np.pi/omg                           # 最高阶模态周期
    #           t_num = int(time_length/period*100.0) + 1

    t_num = num_seg_per_time_window   # 时间窗分段数
```

```
    # 为数据池配置空间
    t_pool = [0.0] * (t_num * num_pool)   # 矢量 t_pool，num_pool*t_num 维
    A_pool = [[0.0 for col in range(num_mode)] for row in
                range(t_num * num_pool)]   # 矩阵 A_pool，num_pool*t_num 行，
num_mode 列

    print "theBridge.omegaList = ", theBridge.omegaList
    print "theBridge.zetaList    = ", theBridge.zetaList

    time_cond = threading.Condition()
    dict_lock = threading.Lock()

    manager = Manager(time_cond, dict_lock)
    monitor = Monitor(time_cond, dict_lock)
    schedul = Schedul(time_cond, dict_lock)
    analyst = Analyst(time_cond, dict_lock)

    manager.start()
    monitor.start()
    schedul.start()
    analyst.start()

    #       ShowAnimate(dict_lock)

    manager.join()
    monitor.join()
    schedul.join()
    analyst.join()


MasterController()
```

8.4 文件四：draw_static.py

```
# coding: utf-8

from matplotlib import pyplot as plt

f = open('output_analyst.txt', 'r')
```

```
data_lines = f.readlines()
num_line = len(data_lines) - 2

t = [0.0 for i in range(num_line)]
A1 = [0.0 for i in range(num_line)]
A2 = [0.0 for i in range(num_line)]
A3 = [0.0 for i in range(num_line)]

for i in range(num_line):
    data_line = data_lines[i + 1]
    (t_, A1_, A2_, A3_) = eval(data_line)
    t[i] = t_
    A1[i] = -A1_
    A2[i] = -A2_
    A3[i] = -A3_
# (t[i], A1[i], A2[i], A3[i]) = eval(data_line)

f.close()
print "num_line =%6i" % num_line
print "(t0, t1) =(%9.3f, %9.3f)" % (t[0], t[num_line - 1])
print t
print A1

# figure()
plt.plot(t, A1, 'r', t, A2, 'b', t, A3, 'k')
plt.xlabel('time(sec)')
plt.ylabel('mode responses(m)')
plt.show()
```

8.5 文件五：draw_animate.py

```
# coding: utf-8

from matplotlib import pyplot as plt
from matplotlib import animation

import numpy as np

ta = list()
```

```
A1 = list()
A2 = list()
A3 = list()

t0_array = list()
t1_array = list()
t2_array = list()
t3_array = list()

v_array = list()
d_array = list()
f_array = list()

num_time = 0
num_vehicle = 0

l_m = 50
l_s = 50

fig = plt.figure()
ax = plt.axes(xlim=(-l_s, l_m + l_s), ylim=(-0.3, 0.3))

line_m_brg, = ax.plot([], [], lw=2)
line_l_brg, = ax.plot([-l_s, 0], [0, 0], lw=2)
line_r_brg, = ax.plot([l_m, l_m + l_s], [0, 0], lw=2)
arr_vehicle = list()   # [ax.annotate("", (0, 0),(0,0), arrowprops=dict(arrowstyle="->")) for i in range(20)]

num_x = 101
x = np.linspace(0, l_m, num_x)


def InitAnimate():
    global num_time, num_vehicle, arr_vehicle
    line_m_brg.set_data([], [])
    return line_m_brg, arr_vehicle


def UpdateAnimate(ii):
    global num_time, num_vehicle, arr_vehicle
```

```
    i = (2 * ii) % num_time    # 取值越大速度越大
    pi = np.pi
    pi1_l = pi / l_m
    pi2_l = pi * 2 / l_m
    pi3_l = pi * 3 / l_m

    mode1 = np.sin(pi1_l * x)
    mode2 = np.sin(pi2_l * x)
    mode3 = np.sin(pi3_l * x)

    t = ta[i]
    a1 = -A1[i]
    a2 = -A2[i]
    a3 = -A3[i]

    y = a1 * mode1 + a2 * mode2 + a3 * mode3    # y = A1*sin(1*pi*x/l) + A2*sin(2*pi*x/l) +
A3*sin(3*pi*x/l) + ...

    line_m_brg.set_data(x, y)

    for idx_v in range(num_vehicle):
        t0 = t0_array[idx_v]
        #             t1 = t1_array[idx_v]
        #             t2 = t2_array[idx_v]
        t3 = t3_array[idx_v]

        v = v_array[idx_v]
        d = d_array[idx_v]
        f = f_array[idx_v]

        if t < t0 or t >= t3:
            arr_vehicle[idx_v].xy = (0, 0)    # arrow head
            arr_vehicle[idx_v].xyann = (0, 0)    # arrow tail
            continue

        if d == "from_left":
            x_pos = -l_s + (t - t0) * v
        else:
            x_pos = l_m + l_s - (t - t0) * v

        if 0 <= x_pos and x_pos < l_m:    # on main bridge
```

```
            ix = np.floor(x_pos / (l_m / (num_x - 1.0)))    # interpolent
            (x0, y0) = (x[ix], y[ix])
            (x1, y1) = (x[ix + 1], y[ix + 1])
            y_pos = ((x_pos - x0) * y1 - (x_pos - x1) * y0) / (x1 - x0)    # interpolent
        else:
            y_pos = 0    # y=0

        arr_vehicle[idx_v].xy = (x_pos, y_pos)    # arrow head
        arr_vehicle[idx_v].xyann = (x_pos, y_pos + 0.1)    # arrow tail

    return line_m_brg, arr_vehicle


# call the animator.    blit=True means only re-draw the parts that have changed.
def ShowAnimate():
    global num_time, num_vehicle, arr_vehicle
    f = open('output_analyst.txt', 'r')

    data_lines = f.readlines()
    num_time = len(data_lines) - 2

    for i in range(num_time):
        data_line = data_lines[i + 1]
        (t_, A1_, A2_, A3_) = eval(data_line)
        ta.append(t_)
        A1.append(A1_)
        A2.append(A2_)
        A3.append(A3_)

    f.close()

    f = open('output_monitor.txt', 'r')

    data_lines = f.readlines()
    num_vehicle = len(data_lines)

    arr_vehicle = [ax.annotate("", (0, 0), (0, 0), arrowprops=dict(arrowstyle="->")) for i in
range(num_vehicle)]
    for i in range(num_vehicle):
        data_line = data_lines[i]
        (name, ts, delay, duration, force, velo, direction, brg_id) = eval(data_line)
```

```
            t0_array.append(ts)
            t1_array.append(ts + delay)
            t2_array.append(ts + delay + duration)
            t3_array.append(ts + delay + duration + delay)

            v_array.append(velo)
            d_array.append(direction)
            f_array.append(force)

    f.close()

    anim = animation.FuncAnimation(fig, UpdateAnimate, interval=5, init_func=InitAnimate,
blit=False)

    plt.show()

ShowAnimate()
```