

Verification and Validation

Tutorial 02


Prof. Americo Cunha Jr

Rio de Janeiro State University – UERJ

americo.cunha@uerj.br


www.americocunha.org



 @AmericoCunhaJr

 @AmericoCunhaJr

 @AmericoCunhaJr

 @AmericoCunhaJr



Verification and Validation (V&V)

- Verification

Are we solving the equation *right*?

- Validation

Are we solving the *right* equation?



G. Iaccarino *Quantification of Uncertainty in Flow Simulations Using Probabilistic Methods*,

VKI Lecture Series, Stanford University, 2008



Verification and Validation (V&V)

- Verification

Are we solving the equation *right*?

It is an exercise in *mathematics*.

- Validation

Are we solving the *right* equation?



G. Iaccarino *Quantification of Uncertainty in Flow Simulations Using Probabilistic Methods*,

VKI Lecture Series, Stanford University, 2008



Verification and Validation (V&V)

- Verification

Are we solving the equation *right*?

It is an exercise in *mathematics*.

- Validation

Are we solving the *right* equation?

It is an exercise in *physics*.



G. Iaccarino *Quantification of Uncertainty in Flow Simulations Using Probabilistic Methods*,

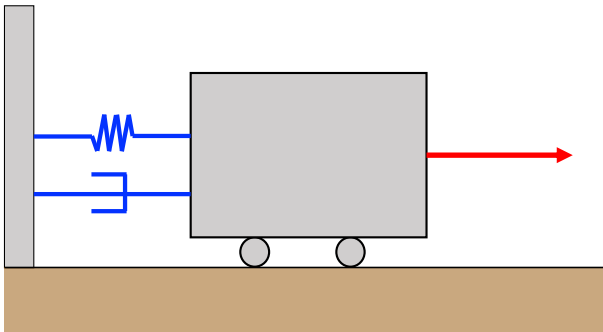
VKI Lecture Series, Stanford University, 2008



V&V for Mass-Spring-Damper Oscillator



Mass-Spring-Damper Oscillator



$$\ddot{x} + 2\xi\omega_n\dot{x} + \omega_n^2x = (f/m)\sin(\omega t)$$

$$\dot{x}(0) = v_0, \quad x(0) = x_0$$

Model equation

$$\ddot{x} + 2\xi\omega_n\dot{x} + \omega_n^2 x = (f/m) \sin(\omega t)$$

$$\underbrace{\begin{bmatrix} \dot{\phi}_1 \\ \dot{\phi}_2 \end{bmatrix}}_{\dot{\phi}} = \underbrace{\begin{bmatrix} 0 & 1 \\ -\omega_n^2 & -2\xi\omega_n \end{bmatrix} \begin{bmatrix} \phi_1 \\ \phi_2 \end{bmatrix} + \begin{bmatrix} 0 \\ (f/m) \sin(\omega t) \end{bmatrix}}_{f(t,\phi)} \quad \Longleftrightarrow$$

where $\phi_1 = x$ and $\phi_2 = \dot{x}$.

Reference solution

The unforced case, that corresponds to $f = 0$, has an analytical solution, which is given by

$$x = A e^{-\xi \omega_n t} \sin(\omega_d t + \phi),$$

where

$$\omega_d = \omega_n \sqrt{1 - \xi^2},$$

$$A = \sqrt{x_0^2 + \left(\frac{v_0 + \xi \omega_n x_0}{\omega_d} \right)^2},$$

$$\phi = \tan^{-1} \left(\frac{x_0 \omega_d}{v_0 + \xi \omega_n x_0} \right).$$

This solution will be used as reference in Verification step.



Numerical method

The initial value problem can be written as

$$\dot{\phi} = f(t, \phi), \quad \phi(0) = \phi_0$$

where $\phi = \begin{bmatrix} x & \dot{x} \end{bmatrix}^T$.

Numerical integration will be done via Explicit Euler method

$$\phi_{n+1} = \phi_n + \Delta t f(t_n, \phi_n)$$

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

where ϕ_n is an approximation for $\phi(t_n)$.



main_verification1.m

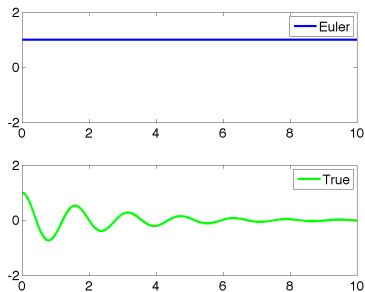
```
1  clc; clear all; close all;
2  m = 1.0; ksi = 0.1; wn = 4.0; f = 0; w = 5.0;
3  x0 = 1.0; v0 = 0.0; t0 = 0.0; t1 = 10.0; N = 3000;
4
5  dphidt=@(t,phi)[0 1; -wn^2 -2*ksi*wn]*phi ...
6      + [0; (f/m)*sin(w*t)];
7
8  [time,phi] = euler(dphidt,[x0;v0],t0,t1,N);
9
10 wd      = wn*sqrt(1-ksi^2);
11 A        = sqrt(x0^2 + ((v0+ksi*wn*x0)/wd)^2);
12 theta    = atan((x0*wd)/(v0+ksi*wn*x0));
13 x_true    = A*exp(-ksi*wn*time).*sin(wd*time+theta);
14
15 subplot(2,1,1)
16 plot(time,phi(1,:), 'b', 'LineWidth',3);
17 legend('Euler')
18 subplot(2,1,2)
19 plot(time,x_true, 'g', 'LineWidth',3);
20 legend('True')
```

euler.m

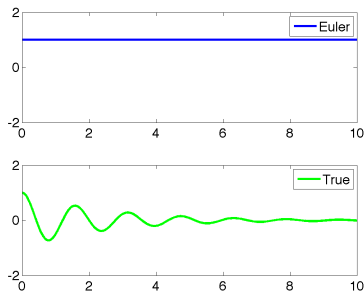
```
1
2 function [time,phi] = euler(rhs,phi0,t0,t1,N)
3
4 dt      = (t1-t0)/N;
5 time    = zeros(1,N+1);
6 phi     = zeros(length(phi0),N+1);
7 phi(:,1) = phi0;
8
9 for n = 1:N
10     time(1,n+1) = time(1,n) + dt;
11     phi(:,n+1) = phi(:,n) + dt*rhs(t0,phi0);
12 end
13
14 return
```



Verification of the equation solution (unforced case)

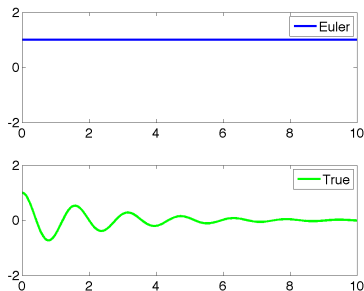


Verification of the equation solution (unforced case)



Something is wrong!
(numeric different from analytic)

Verification of the equation solution (unforced case)



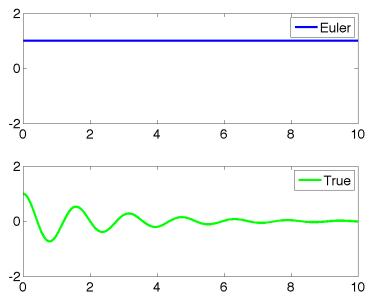
Something is wrong!

(numeric different from analytic)

There is a **bug** in euler.m routine:



Verification of the equation solution (unforced case)



Something is wrong!

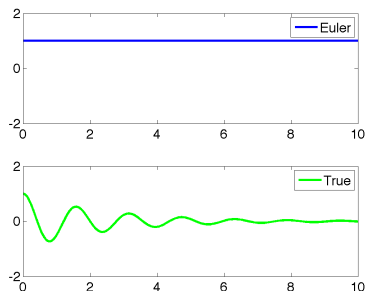
(numeric different from analytic)

There is a **bug** in euler.m routine:

```
phi(:,n+1) = phi(:,n) + dt*rhs(t0,phi0);
```



Verification of the equation solution (unforced case)



Something is wrong!

(numeric different from analytic)

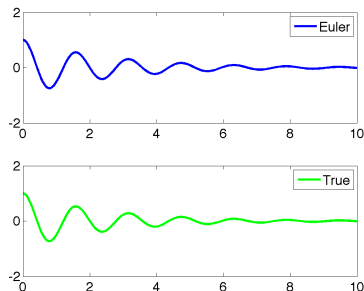
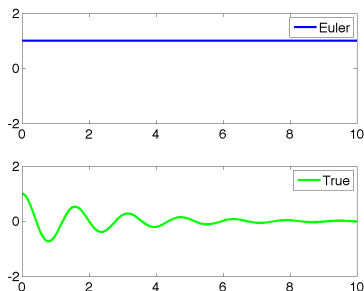
There is a **bug** in euler.m routine:

```
phi(:,n+1) = phi(:,n) + dt*rhs(t0,phi0);
```

```
phi(:,n+1) = phi(:,n) + dt*rhs(t0,phi(:,n));
```



Verification of the equation solution (unforced case)



Something is wrong!

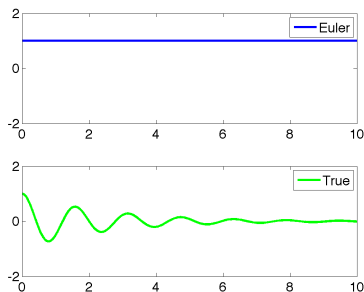
(numeric different from analytic)

There is a **bug** in euler.m routine:

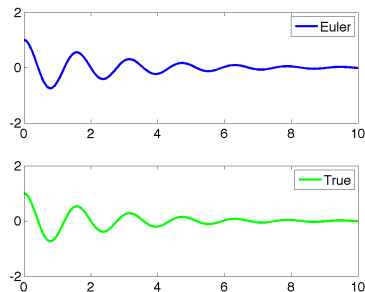
```
phi(:,n+1) = phi(:,n) + dt*rhs(t0,phi0);  
phi(:,n+1) = phi(:,n) + dt*rhs(t0,phi(:,n));
```



Verification of the equation solution (unforced case)



Something is wrong!
(numeric different from analytic)



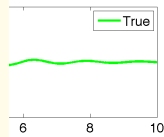
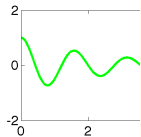
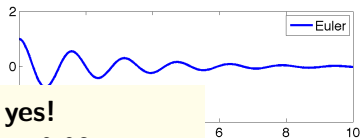
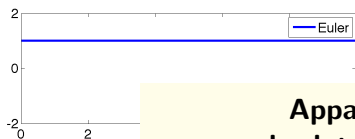
Is the code working fine?!

There is a **bug** in euler.m routine:

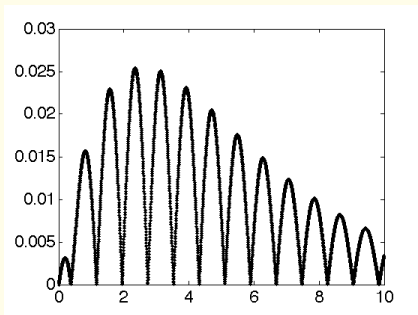
```
phi(:,n+1) = phi(:,n) + dt*rhs(t0,phi0);  
phi(:,n+1) = phi(:,n) + dt*rhs(t0,phi(:,n));
```



Verification of the equation solution (unforced case)



Apparently yes!
absolute error ≤ 0.03



Something
(numeric differ

king fine?!

There is a **bu**
phi(:,n+1)
phi(:,n+1)

phi(:,n+1);



main_verification2.m

```
1  clc; clear all; close all;
2  m = 1.0; ksi = 0.1; wn = 4.0; f = 10.0; w = 5.0;
3  x0 = 1.0; v0 = 0.0; t0 = 0.0; t1 = 10.0; N = 3000;
4
5  dphidt=@(t,phi)[0 1; -wn^2 -2*ksi*wn]*phi ...
6                + [0; (f/m)*sin(w*t)];
7
8  [time,phi] = euler(dphidt,[x0;v0],t0,t1,N);
9
10 plot(time,phi(1,:), 'LineWidth',3);
```



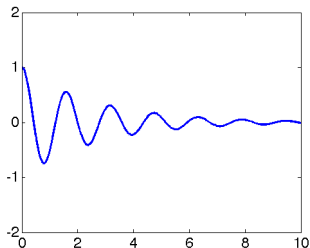
Verification of the equation solution (forced case)

Imagine that analytical solution is not known.



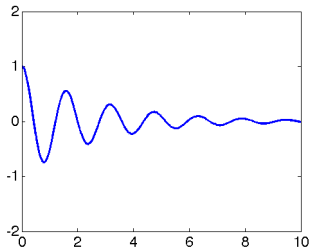
Verification of the equation solution (forced case)

Imagine that analytical solution is not known.



Verification of the equation solution (forced case)

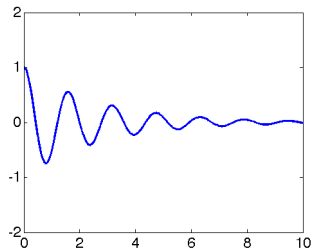
Imagine that analytical solution is not known.



Something is wrong!
(incompatible with harmonic forcing)

Verification of the equation solution (forced case)

Imagine that analytical solution is not known.



Something is wrong!

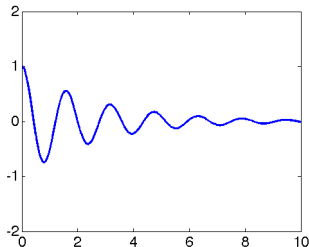
(incompatible with harmonic forcing)

The routine euler.m still has **bug(s)**:



Verification of the equation solution (forced case)

Imagine that analytical solution is not known.



Something is wrong!

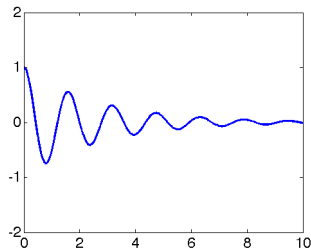
(incompatible with harmonic forcing)

The routine euler.m still has **bug(s)**:

```
phi(:,n+1) = phi(:,n) + dt*rhs(t0,phi(:,n));
```

Verification of the equation solution (forced case)

Imagine that analytical solution is not known.



Something is wrong!

(incompatible with harmonic forcing)

The routine euler.m still has **bug(s)**:

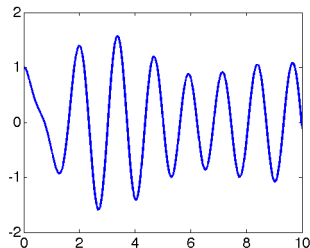
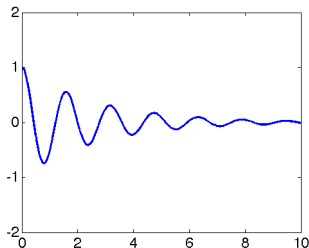
```
phi(:,n+1) = phi(:,n) + dt*rhs(t0,phi(:,n));
```

```
phi(:,n+1) = phi(:,n) + dt*rhs(time(1,n),phi(:,n));
```



Verification of the equation solution (forced case)

Imagine that analytical solution is not known.



Something is wrong!

(incompatible with harmonic forcing)

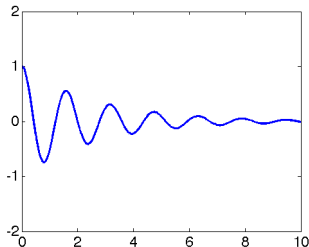
The routine euler.m still has **bug(s)**:

```
phi(:,n+1) = phi(:,n) + dt*rhs(t0,phi(:,n));
```

```
phi(:,n+1) = phi(:,n) + dt*rhs(time(1,n),phi(:,n));
```

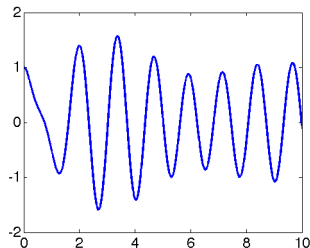
Verification of the equation solution (forced case)

Imagine that analytical solution is not known.



Something is wrong!

(incompatible with harmonic forcing)



Is this the correct response?

The routine euler.m still has **bug(s)**:

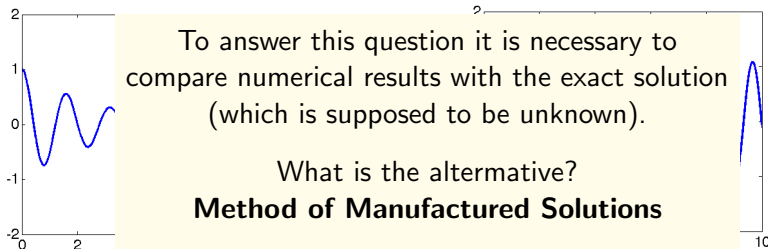
```
phi(:,n+1) = phi(:,n) + dt*rhs(t0,phi(:,n));
```

```
phi(:,n+1) = phi(:,n) + dt*rhs(time(1,n),phi(:,n));
```



Verification of the equation solution (forced case)

Imagine that analytical solution is not known.



Something is wrong!

(incompatible with harmonic forcing)

Is this the correct response?

The routine euler.m still has **bug(s)**:

```
phi(:,n+1) = phi(:,n) + dt*rhs(t0,phi(:,n));
```

```
phi(:,n+1) = phi(:,n) + dt*rhs(time(1,n),phi(:,n));
```



Method of Manufactured Solutions

The idea is to construct (manufacture) an initial value problem (IVP) in which the solution is known, and use it to test the numerical integrator functionality.

1. Choose the form of model equations

$$\dot{\phi} = f(t, \phi), \quad \phi(0) = \phi_0 \quad (\star)$$

2. Define a manufactured solution Θ such that $\Theta(0) = \phi_0$, which does not verify (\star) , i.e. $\dot{\Theta} \neq f(t, \Theta)$.
3. Compute the residue function $\mathcal{R}(t) := \dot{\Theta} - f(t, \Theta) \neq 0$.
4. Define the manufactured IVP

$$\dot{\Theta} = f(t, \Theta) + \mathcal{R}(t), \quad \Theta(0) = \phi_0,$$

which is verified by the manufactured solution Θ .



Method of Manufactured Solutions

Example: (Forced MSD Oscillator)

Take as manufactured solution $\Theta = \begin{bmatrix} \cos t & \sin t \end{bmatrix}^T$, which satisfies the initial conditions.

This is not a solution for the forced oscillator, since

$$\underbrace{\begin{bmatrix} -\sin t \\ \cos t \end{bmatrix}}_{\dot{\Theta}} \neq \underbrace{\begin{bmatrix} 0 & 1 \\ -\omega_n^2 & -2\xi\omega_n \end{bmatrix} \begin{bmatrix} \cos t \\ \sin t \end{bmatrix} + \begin{bmatrix} 0 \\ (f/m) \sin(\omega t) \end{bmatrix}}_{f(t,\Theta)}.$$

Then, the residual function is

$$\mathcal{R}(t) = \begin{bmatrix} -2 \sin t \\ \cos t + 2\xi\omega_n \sin t + \omega_n^2 \cos t - (f/m) \sin(\omega t) \end{bmatrix}.$$



Method of Manufactured Solutions

The manufactured initial value problem is

$$\begin{bmatrix} \dot{\Theta}_1 \\ \dot{\Theta}_2 \end{bmatrix} = \begin{bmatrix} \Theta_2 - 2 \sin t \\ -\omega_n^2 \Theta_1 - 2\xi\omega_n \Theta_2 + \cos t + 2\xi\omega_n \sin t + \omega_n^2 \cos t \end{bmatrix},$$

where $\Theta_1(0) = 1$ and $\Theta_2 = 0$. Indeed,

$$\begin{bmatrix} \Theta_1 \\ \Theta_2 \end{bmatrix} = \begin{bmatrix} \cos t \\ \sin t \end{bmatrix}$$

is a solution. **Verify yourself!**

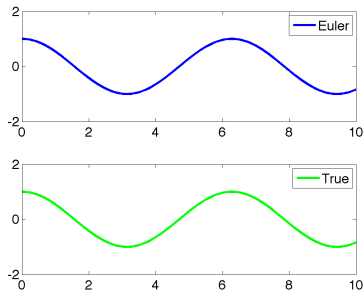


main_verification3.m

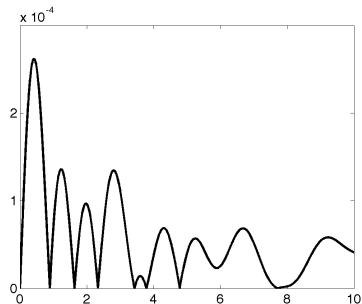
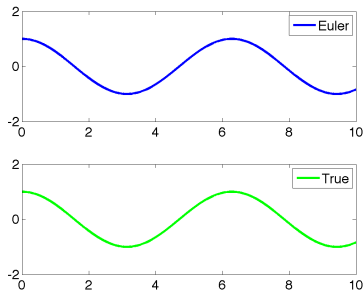
```
1  clc; clear all; close all;
2  m = 1.0; ksi = 0.1; wn = 4.0; f = 10.0; w = 5.0;
3  x0 = 1.0; v0 = 0.0; t0 = 0.0; t1 = 10.0; N = 5000;
4
5  dphidt=@(t,phi)[0 1; -wn^2 -2*ksi*wn]*phi ...
6                + [-2*sin(t); ...
7                cos(t)+2*ksi*wn*sin(t)+wn^2*cos(t)];
8
9  [time,phi] = euler(dphidt,[x0;v0],t0,t1,N);
10
11 x_true = cos(time);
12
13 subplot(2,1,1)
14 plot(time,phi(1,:), 'b', 'LineWidth',3);
15 legend('Euler')
16 subplot(2,1,2)
17 plot(time,x_true, 'g', 'LineWidth',3);
18 legend('True')
```



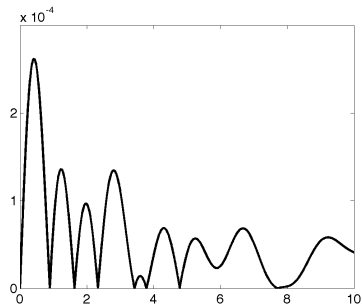
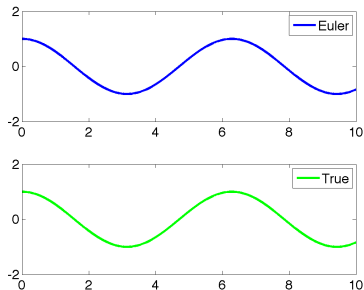
Verification of the equation solution (manufactured IVP)



Verification of the equation solution (manufactured IVP)

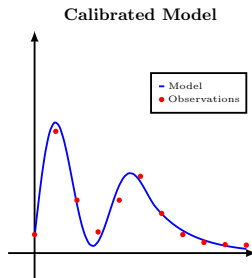
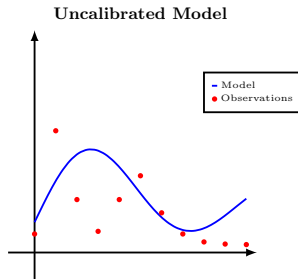


Verification of the equation solution (manufactured IVP)



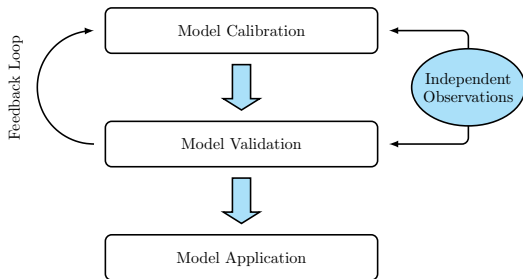
Euler method is working fine!

Model Calibration and Model Validation

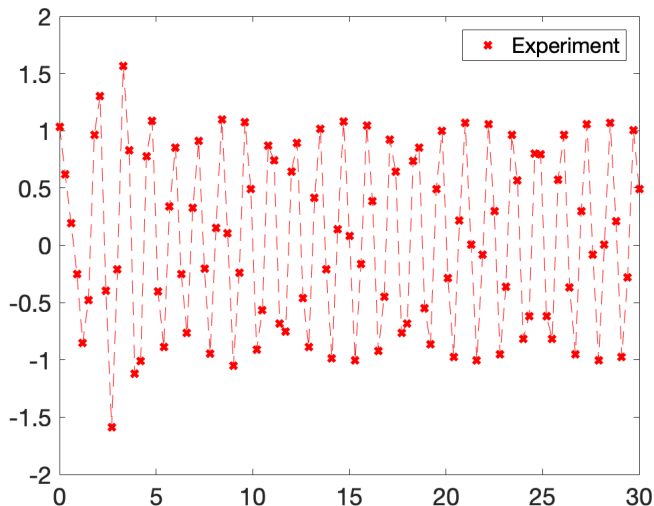


* Left pictures prepared by Michel Tosin.

Model Calibration and Validation



Validation case 1: experimental data set

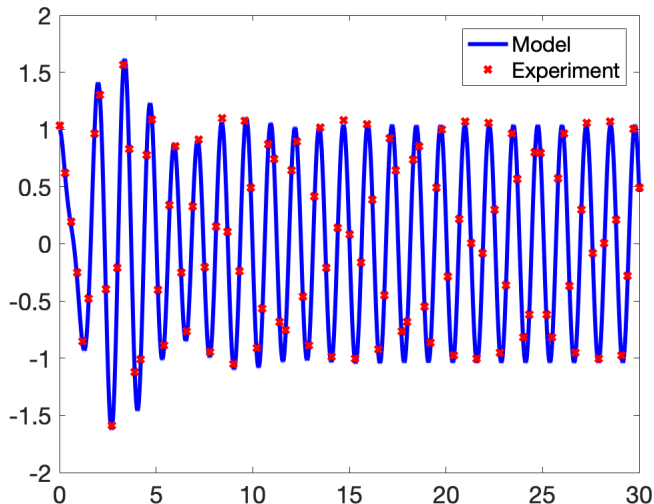


main_validation1.m

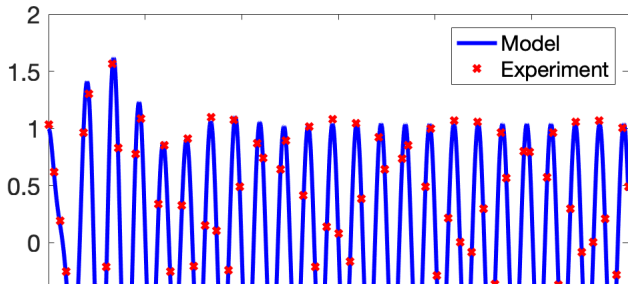
```
1  clc; clear; close all;
2  m = 1.0; ksi = 0.1; wn = 4.0; f = 10.0; w = 5.0;
3  x0 = 1.0; v0 = 0.0; t0 = 0.0; t1 = 30.0; N = 5000;
4
5  dphidt=@(t,phi)[0 1; -wn^2 -2*ksi*wn]*phi ...
6                + [0; (f/m)*sin(w*t)];
7
8  [t,phi] = euler(dphidt,[x0;v0],t0,t1,N);
9
10 t_exp = t(1:50:end);
11 x_exp = phi(1,1:50:end) + 0.01*randn;
12
13 plot(t,phi(1,:), 'b', t_exp, x_exp, 'xr', 'LineWidth', 3);
14 axis([t0 t1 -2 2])
15 legend('Model', 'Experiment')
```



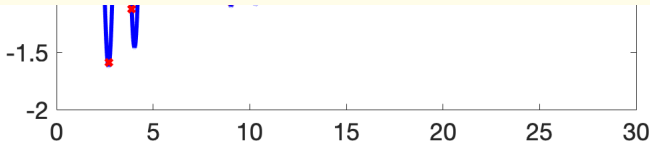
Validation case 1: predictions and observations



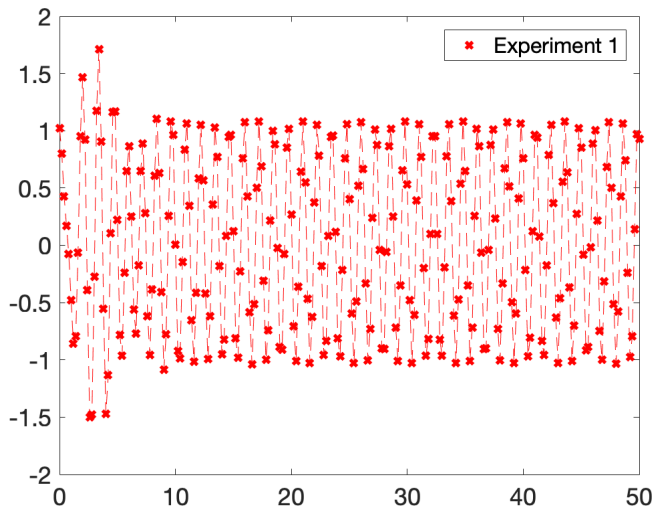
Validation case 1: predictions and observations



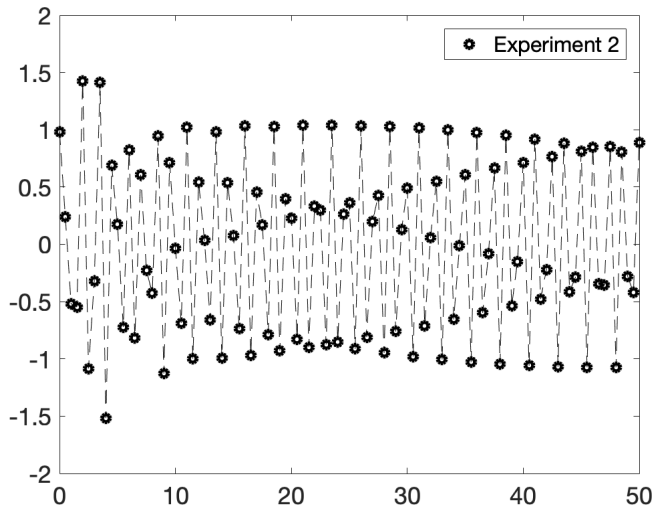
Good agreement between experiment and simulation!
Validated Model!



Validation case 2: First experimental data set



Validation case 2: Second experimental data set

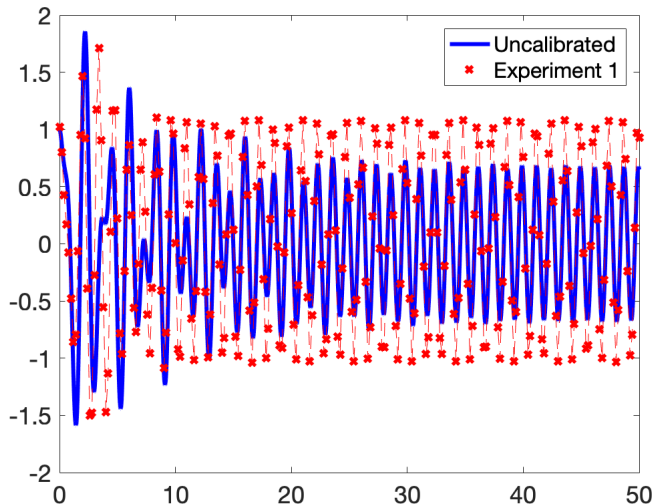


main_validation2.m (1/3)

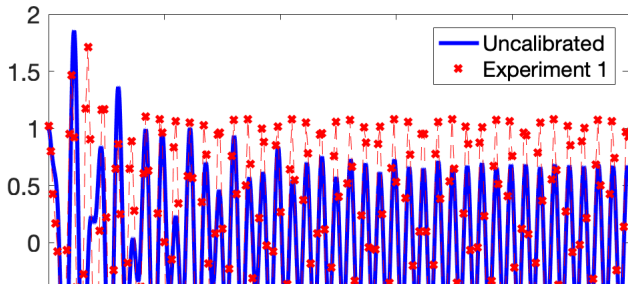
```
1  clc; clear; close all;
2  m = 1.0; ksi = 0.1; wn = 4.0; f = 10.0; w = 5.0;
3  x0 = 1.0; v0 = 0.0; t0 = 0.0; t1 = 50.0; N = 5000;
4
5  dphidt=@(t,phi)[0 1; -wn^2 -2*ksi*wn]*phi + [0; (f/m)*sin(w*t)];
6  [t_ref,phi_ref] = euler(dphidt,[x0;v0],t0,t1,N);
7  t_exp1 = t_ref(1:20:end); x_exp1 = phi_ref(1,1:20:end) + 0.05*randn;
8  t_exp2 = t_ref(1:50:end); x_exp2 = phi_ref(1,1:50:end) + 0.1*randn;
9
10 figure(1)
11 plot(t_exp1,x_exp1,'xr','LineWidth',3);
12 hold on
13 plot(t_exp1,x_exp1,'--r','LineWidth',0.3);
14 hold off
15 axis([t0 t1 -2 2]); set(gca,'fontsize',18); legend('Experiment 1');
16
17 figure(2)
18 plot(t_exp2,x_exp2,'ok','LineWidth',3);
19 hold on
20 plot(t_exp2,x_exp2,'--k','LineWidth',0.3);
21 hold off
22 axis([t0 t1 -2 2]); set(gca,'fontsize',18); legend('Experiment 2');
```



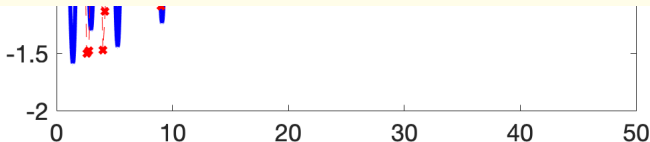
Validation case 2: predictions and observations



Validation case 2: predictions and observations



Experiment and simulation are not in agreement!
Invalid Model!

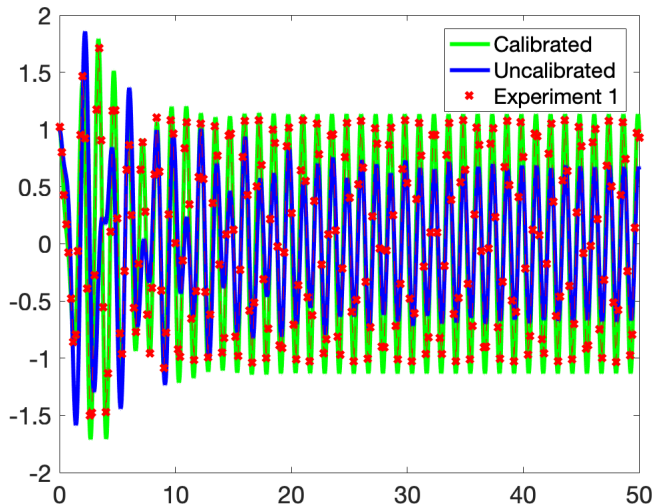


main_validation2.m (2/3)

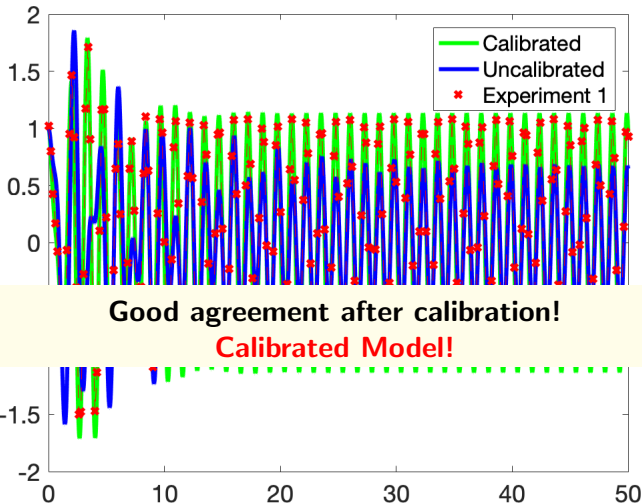
```
1 m = 1.0; ksi = 0.05; wn = 3.2; f = 10.0; w = 5.0;
2 x0 = 1.0; v0 = 0.0;
3
4 dphidt=@(t,phi)[0 1; -wn^2 -2*ksi*wn]*phi + [0; (f/m)*sin(w*t)];
5 [t_uncal,phi_uncal] = euler(dphidt,[x0;v0],t0,t1,N);
6
7 figure(3)
8 plot(t_uncal,phi_uncal(1,:), 'b', 'LineWidth',3);
9 hold on
10 plot(t_exp1,x_exp1, 'xr', 'LineWidth',3);
11 plot(t_exp1,x_exp1, '--r', 'LineWidth',0.3);
12 hold off
13 axis([t0 t1 -2 2])
14 set(gca, 'fontsize',18)
15 legend('Uncalibrated','Experiment 1')
```



Validation case 2: model calibration



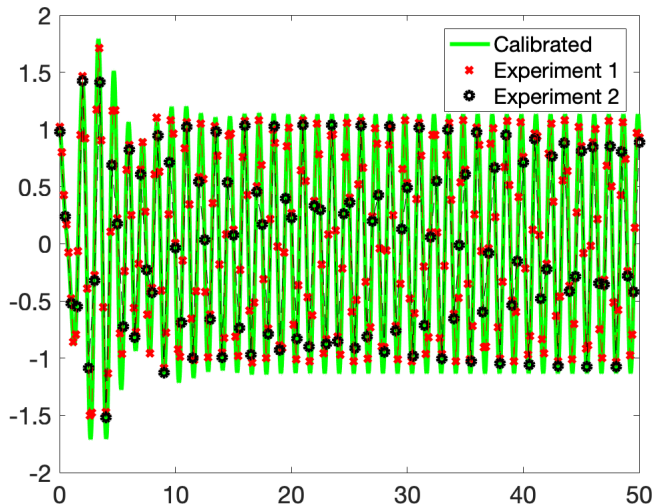
Validation case 2: model calibration



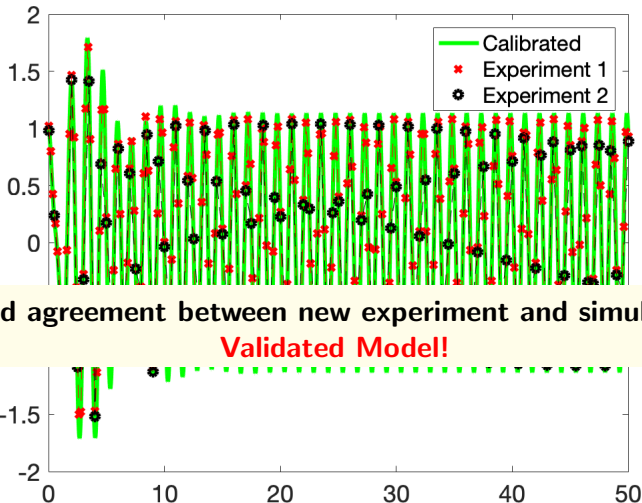
main_validation2.m (3/3)

```
1 m = 1.0; ksi = 0.095; wn = 4.08; f = 10.0; w = 5.0;
2 x0 = 1.0; v0 = 0.0;
3
4 dphidt=@(t,phi)[0 1; -wn^2 -2*ksi*wn]*phi + [0; (f/m)*sin(w*t)];
5 [t_cal,phi_cal] = euler(dphidt,[x0;v0],t0,t1,N);
6
7 figure(4)
8 plot(t_cal,phi_cal(1,:), 'g',t_uncal,phi_uncal(1,:), 'b', 'LineWidth',3);
9 hold on
10 plot(t_exp1,x_exp1, 'xr', 'LineWidth',3);
11 plot(t_exp1,x_exp1, '--r', 'LineWidth',0.3);
12 hold off
13 axis([t0 t1 -2 2]); set(gca, 'fontSize',18);
14 legend('Calibrated','Uncalibrated','Experiment 1')
15
16 figure(5)
17 plot(t_cal,phi_cal(1,:), 'g', 'LineWidth',3);
18 hold on
19 plot(t_exp1,x_exp1, 'xr', 'LineWidth',3);
20 plot(t_exp2,x_exp2, 'ok', 'LineWidth',3);
21 plot(t_exp1,x_exp1, '--r', 'LineWidth',0.3);
22 plot(t_exp2,x_exp2, '--k', 'LineWidth',0.3);
23 hold off
24 axis([t0 t1 -2 2]); set(gca, 'fontSize',18);
25 legend('Calibrated','Experiment 1','Experiment 2')
```

Validation case 2: calibrated model and new observations



Validation case 2: calibrated model and new observations



Good agreement between new experiment and simulation!
Validated Model!

References



C. J. Roy and W. L. Oberkampf, *A comprehensive framework for verification, validation, and uncertainty quantification in scientific computing*. **Computer Methods in Applied Mechanics and Engineering**, 200: 2131–2144, 2011.



C. J. Roy, *Review of code and solution verification procedures for computational simulation*. **Journal of Computational Physics**, 205: 131–156, 2005.



W. L. Oberkampf, T. Trucano and C. Hirsch, *Verification, validation, and predictive capability in computational engineering and physics*. **Applied Mechanics Reviews**, 57: 345–384, 2004.



W. L. Oberkampf and C. J. Roy, **Verification and Validation in Scientific Computing**. Cambridge University Press, 1st edition, 2010.

