PECOS

Predictive Engineering and Computational Sciences

# MASA: Manufactured Analytical Solutions Abstractions Library

Nicholas Malaya

Institute for Computational Engineering and Sciences
The University of Texas at Austin

June 23rd, 2010

# Outline


1. Introduction to Verification
2. Manufactured Solutions
3. Code Design
4. Conclusions & Future Work

# PSAAP Goals

## Predictive Science Academic Alliance Program

- Focus on multi-scale, multidisciplinary, unclassified applications of NNSA interest
- Demonstrate validated simulation capability for prediction
- Produce significant science / engineering results
- Produce new methodologies
  - Verification
  - Validation
  - Uncertainty Quantification
  - Tighter integration of experiment & simulation
- Improve quantity & quality of tools and algorithms

# Verification

- Verification is the act of proving or disproving the correctness of algorithms underlying a system.
- Essentially, we are testing if we have correctly instantiated mathematical equations in our code.

### Many uses of verification at PECOS

- Scientific Software center
- Generating truth data
- Required before validation, calibration or uncertainty quantification

- An unverified code is not a functioning code.
- How do we check our code is functioning properly?

# Outline

1 Introduction to Verification

2 Manufactured Solutions

3 Code Design

4 Conclusions & Future Work

# Manufactured Solution

- For the vast majority of our problems, we do not possess analytical solutions.
- Getting around a lack of analytical solutions is the purpose of manufactured solutions.

## Art of the MMS

- An artificially generated (manufactured) analytical solution
- Commonly developed using trigonometric functions or polynomials
- Need not be physical – only testing fidelity of mathematics

# Generating Manufactured Solutions is Straightforward

## Method

- cast equation as an operator

$$O(u) = 0 \tag{1}$$

- adding the source terms to the RHS

$$O(u) = S_u \tag{2}$$

- inserting a generated analytical solution

$$u\left(x,y\right) = u_0 + u_x \sin\left(\frac{a_{ux}\pi x}{L}\right) + u_y \cos\left(\frac{a_{uy}\pi y}{L}\right) \tag{3}$$

- apply operator to the analytical solution to obtain the source term
- Essentially, we have verified the operator.

## Toy Problem

Solve:

$$u''(x) = 0 \tag{4}$$

Add source term to RHS:

$$\frac{d^2}{dx^2}(u(x)) = S_u \tag{5}$$

Assume a solution of the form,

$$u(x) = ax^5 + bx^3 + cx + d \tag{6}$$

Apply operator to solution.

$$\frac{d^2}{dx^2}(ax^5 + bx^3 + cx + d) = S_u \tag{7}$$

$$20ax^3 + 6bx = S_u \tag{8}$$

# Manufactured Solutions using Maple, Mathematica

```
> # This program calculates the source term Q for the 1D steady
  temperature equation with constant K
> # - nabla (K nabla T) = 0, T=T(x),
> # K = k0
> # so the modified equation:
  #  - nabla (K nabla T) = Q has analytical manufactured
  solution:
  # T_an := cos(A_x*x)

> restart;
> with(CodeGeneration) :
> alias(T = T(x)) :
> alias(T_an = T_an(x)) :
> alias(Q = Q(x)) :
> alias(k = k(T)) :

> # K is constant
> k := k_0 :

> # 1D steady temperature equation
> -Diff(k·Diff(T, x), x) = 0;
```

$$-\left(\frac{\partial}{\partial x}\left(k\_0\left(\frac{\partial}{\partial x}T\right)\right)\right)=0 \quad \textbf{(1)}$$

```
> # Defining operator L(T), for manufacturing the source term Q
> L := -diff(k·diff(T, x), x) :

> # Choosing an analytical solution for T
> T_an := cos(A_x·x);
```

$$T\_an := \cos(A\_x\,x) \quad \textbf{(2)}$$

```
> # Applying operator L on T_an, in order to obtain Q
> Q := algsubs(T = T_an, L) :
> Q := simplify(Q, trig) :
> Q_T := sort(Q)
```

$$Q\_T := A\_x^2\,k\_0\cos(A\_x\,x) \quad \textbf{(3)}$$

```
> # Calculate nabla T_an
> gradT_an[1] := sort(diff(T_an, x));
```

$$gradT\_an_1 := -A\_x\sin(A\_x\,x) \quad \textbf{(4)}$$

```
> # Writing source term Q, manufactured solution T_an and vector
  flux gradT_an as a procedure
> SourceQ := proc(x, A_x, k_0)
  local Q_T, T_an, gradT_an;
```

## method:

- This method can be extended to complex systems of equations (Navier-Stokes)
- non-trivial to solve by hand - use symbolic manipulation software packages
- Assumed form of solution is always a ‘guess’

### The 1D Euler equations:

$$\frac{\partial(\rho)}{\partial t} + \frac{\partial(\rho u)}{\partial x} = 0, \tag{9}$$

$$\frac{\partial(\rho u)}{\partial t} + \frac{\partial(\rho u^2 + p)}{\partial x} = 0, \tag{10}$$

$$\frac{\partial(\rho e_t)}{\partial t} + \frac{\partial(\rho u e_t + p u)}{\partial x} = 0, \tag{11}$$

For a calorically perfect gas, the Euler equations are closed with two relations for energy:

$$e = \frac{1}{\gamma - 1} RT, \tag{12}$$

$$e_t = e + \frac{u^2}{2}, \tag{13}$$

and with the ideal gas equation of state:

$$p = \rho RT. \tag{14}$$

## Manufacturing a solution:

The manufactured analytical solution for for each one of the variables in Euler equations are:

$$\begin{aligned}
\rho\left(x,y\right) &= \rho_0 + \rho_x \sin\left(\frac{a_{\rho x}\pi x}{L}\right), \\
u\left(x,y\right) &= u_0 + u_x \sin\left(\frac{a_{ux}\pi x}{L}\right), \\
p\left(x,y\right) &= p_0 + p_x \cos\left(\frac{a_{px}\pi x}{L}\right).
\end{aligned} \tag{15}$$

Analytically differentiating Equation for $\rho$ and $u$ gives the source term $Q_\rho$:

$$\begin{aligned}
Q_\rho = \frac{a_{\rho x}\pi\rho_x}{L}\cos\left(\frac{a_{\rho x}\pi x}{L}\right)\left[u_0 + u_x\sin\left(\frac{a_{ux}\pi x}{L}\right)\right] + \\
+ \frac{a_{ux}\pi u_x}{L}\cos\left(\frac{a_{ux}\pi x}{L}\right)\left[\rho_0 + \rho_x\sin\left(\frac{a_{\rho x}\pi x}{L}\right)\right].
\end{aligned} \tag{16}$$

## Euler Continued:

Likewise for the u component of velocity source term, $Q_u$,

$$\begin{aligned}
Q_u = & \frac{a_{\rho x}\pi\rho_x}{L}\cos\left(\frac{a_{\rho x}\pi x}{L}\right)\left[u_0 + u_x\sin\left(\frac{a_{ux}\pi x}{L}\right)\right]^2 + \\
& - \frac{a_{px}\pi p_x}{L}\sin\left(\frac{a_{px}\pi x}{L}\right) + \\
& + \frac{2a_{ux}\pi u_x}{L}\cos\left(\frac{a_{ux}\pi x}{L}\right)\left[\rho_0 + \rho_x\sin\left(\frac{a_{\rho x}\pi x}{L}\right)\right]\left[u_0 + u_x\sin\left(\frac{a_{ux}\pi x}{L}\right)\right].
\end{aligned} \tag{17}$$

and, the source term $Q_e$ is:

$$\begin{aligned}
Q_e = & \frac{a_{\rho x}\pi\rho_x}{2L}\cos\left(\frac{a_{\rho x}\pi x}{L}\right)\left[u_0 + u_x\sin\left(\frac{a_{ux}\pi x}{L}\right)\right]^3 + \\
& - \frac{a_{px}\pi p_x}{L}\frac{\gamma}{\gamma-1}\sin\left(\frac{a_{px}\pi x}{L}\right)\left[u_0 + u_x\sin\left(\frac{a_{ux}\pi x}{L}\right)\right] + \\
& + \frac{a_{ux}\pi u_x}{L}\frac{\gamma}{\gamma-1}\cos\left(\frac{a_{ux}\pi x}{L}\right)\left[p_0 + p_x\cos\left(\frac{a_{px}\pi x}{L}\right)\right] + \\
& + \frac{3a_{ux}\pi u_x}{2L}\cos\left(\frac{a_{ux}\pi x}{L}\right)\left[\rho_0 + \rho_x\sin\left(\frac{a_{\rho x}\pi x}{L}\right)\right]\left[u_0 + u_x\sin\left(\frac{a_{ux}\pi x}{L}\right)\right]^2.
\end{aligned} \tag{18}$$

# This can get complicated

## Source Term: 2d Navier-Stokes

$$
\begin{aligned}
Q_u = & -\frac{a_{px}\pi p_x}{L}\sin\left(\frac{a_{px}\pi x}{L}\right) + \\
& + \frac{a_{\rho x}\pi\rho_x}{L}\cos\left(\frac{a_{\rho x}\pi x}{L}\right)\left(u_0 + u_x\sin\left(\frac{a_{ux}\pi x}{L}\right) + u_y\cos\left(\frac{a_{uy}\pi y}{L}\right) + u_z\cos\left(\frac{a_{uz}\pi z}{L}\right)\right)^2 + \\
& - \frac{a_{\rho y}\pi\rho_y}{L}\sin\left(\frac{a_{\rho y}\pi y}{L}\right)\left(v_0 + v_x\cos\left(\frac{a_{vx}\pi x}{L}\right) + v_y\sin\left(\frac{a_{vy}\pi y}{L}\right) + v_z\sin\left(\frac{a_{vz}\pi z}{L}\right)\right)\bigg(u_0 + u_x\sin\left(\frac{a_{ux}\pi x}{L}\right) + u_y\cos \cdots \\
& + \frac{a_{\rho z}\pi\rho_z}{L}\cos\left(\frac{a_{\rho z}\pi z}{L}\right)\left(w_0 + w_x\sin\left(\frac{a_{wx}\pi x}{L}\right) + w_y\sin\left(\frac{a_{wy}\pi y}{L}\right) + w_z\cos\left(\frac{a_{wz}\pi z}{L}\right)\right)\bigg(u_0 + u_x\sin\left(\frac{a_{ux}\pi x}{L}\right) + u_y\cos \cdots \\
& + \frac{2a_{ux}\pi u_x}{L}\cos\left(\frac{a_{ux}\pi x}{L}\right)\left(\rho_0 + \rho_x\sin\left(\frac{a_{\rho x}\pi x}{L}\right) + \rho_y\cos\left(\frac{a_{\rho y}\pi y}{L}\right) + \rho_z\sin\left(\frac{a_{\rho z}\pi z}{L}\right)\right)\bigg(u_0 + u_x\sin\left(\frac{a_{ux}\pi x}{L}\right) + u_y\cos \cdots \\
& - \frac{a_{uy}\pi u_y}{L}\sin\left(\frac{a_{uy}\pi y}{L}\right)\left(\rho_0 + \rho_x\sin\left(\frac{a_{\rho x}\pi x}{L}\right) + \rho_y\cos\left(\frac{a_{\rho y}\pi y}{L}\right) + \rho_z\sin\left(\frac{a_{\rho z}\pi z}{L}\right)\right)\bigg(v_0 + v_x\cos\left(\frac{a_{vx}\pi x}{L}\right) + v_y\sin \cdots \\
& - \frac{a_{uz}\pi u_z}{L}\sin\left(\frac{a_{uz}\pi z}{L}\right)\left(\rho_0 + \rho_x\sin\left(\frac{a_{\rho x}\pi x}{L}\right) + \rho_y\cos\left(\frac{a_{\rho y}\pi y}{L}\right) + \rho_z\sin\left(\frac{a_{\rho z}\pi z}{L}\right)\right)\bigg(w_0 + w_x\sin\left(\frac{a_{wx}\pi x}{L}\right) + w_y\sin \cdots \\
& + \frac{a_{vy}\pi v_y}{L}\cos\left(\frac{a_{vy}\pi y}{L}\right)\left(\rho_0 + \rho_x\sin\left(\frac{a_{\rho x}\pi x}{L}\right) + \rho_y\cos\left(\frac{a_{\rho y}\pi y}{L}\right) + \rho_z\sin\left(\frac{a_{\rho z}\pi z}{L}\right)\right)\bigg(u_0 + u_x\sin\left(\frac{a_{ux}\pi x}{L}\right) + u_y\cos \cdots \\
& - \frac{a_{wz}\pi w_z}{L}\sin\left(\frac{a_{wz}\pi z}{L}\right)\left(\rho_0 + \rho_x\sin\left(\frac{a_{\rho x}\pi x}{L}\right) + \rho_y\cos\left(\frac{a_{\rho y}\pi y}{L}\right) + \rho_z\sin\left(\frac{a_{\rho z}\pi z}{L}\right)\right)\bigg(u_0 + u_x\sin\left(\frac{a_{ux}\pi x}{L}\right) + u_y\cos \cdots \\
& + \frac{4a_{ux}^2\pi^2\mu u_x}{3L^2}\sin\left(\frac{a_{ux}\pi x}{L}\right) + \frac{a_{uy}^2\pi^2\mu u_y}{L^2}\cos\left(\frac{a_{uy}\pi y}{L}\right) + \frac{a_{uz}^2\pi^2\mu u_z}{L^2}\cos\left(\frac{a_{uz}\pi z}{L}\right)
\end{aligned}
$$

# Manufactured Solutions Generated

## Kemelli generated solutions

- Heat Equation
  - steady / unsteady
  - constant / variable material properties ($\rho$, $c_p$, k)
  - 1d, 2d, 3d
- Euler Equations
  - 1d, 2d, 3d
- Compressible Navier-Stokes Equations
  - 2d, 3d

Many other equations possible: Burgers, Shock tubes, chemistry, etc.

# Outline


1. Introduction to Verification
2. Manufactured Solutions
3. Code Design
4. Conclusions & Future Work

# Enter MASA

## Manufactured Analytical Software Abstraction Library

- central repository for various manufactured solutions
- focus all PECOS MMS efforts on single library for support and standard API

Intended for use with various internal software projects:

- FIN-S
- Suzerain
- Thermocouple
- etc.

# Requirements Documentation

- C++, C, Fortran90 bindings
- Supports gnu, intel compilers (maybe more in the future)
- Meet or exceed all PECOS software standards
- Provide standardized interface for all MMS.
- Targeting LGPL release

Unlike many other applications, performance is not terribly important here.

# Example C++ API

## Intializers

- int masa_init(string, string)
- int masa_init_param()
- int masa_set_param(string,double)
- int masa_get_param(string,*double)
- int masa_select_mms(string)
- int masa_curr_mms(*string)

- All subroutines return integers for error conditions.
- C interface is identical, but with a cmasa_ affix
- Fortran interface has error integer passed at the end

# Example C++ API

## Evaluate

- int masa_eval_t_source (double,double*);
- int masa_eval_u_source (double,double*);
- int masa_eval_e_source (double,double*);
- int masa_eval_rho_source(double,double*);

- int masa_eval_t_an (double,double*);
- int masa_eval_u_an (double,double*);
- int masa_eval_p_an (double,double*);
- int masa_eval_rho_an (double,double*);

Higher dimensional examples will have v,w and require more input.

# C++ Example: Euler Equations

```
#include <masa.h>
using namespace MASA;

int main()
{
  double tempx,ufield,efield,rho,u_an,p_an,rho_an;
  double lx =  1;
  int    nx = 10;
  double dx = double(lx/nx);

  masa_init("euler-example","euler_1d");
  masa_init_param();
  masa_set_param("rho_x",1.4);
  masa_set_param("p_0",  .82);

  masa_sanity_check();

  for(int i=0;i<nx;i++)
     {
      tempx=i*dx;

      masa_eval_u_source  (tempx,&ufield);
      masa_eval_e_source  (tempx,&efield);
      masa_eval_rho_source(tempx,&rho);

      masa_eval_u_an      (tempx,&u_an);
      masa_eval_p_an      (tempx,&p_an);
      masa_eval_rho_an    (tempx,&rho_an);
    }
}// end program
```

# C Example: Heat Equation

```
#include <cmasa.h>
#include <stdio.h>

int main()
{
  double sol,x,an;
  int    nx = 10;
  double lx =  1;
  double dx = double(lx/nx);

  cmasa_init("nick","heateq_1d_steady_const"); // char* here
  cmasa_init_param();
  cmasa_sanity_check();

  for(int i=0;i<nx;i++)
     {
      x=i*dx;
      cmasa_eval_t_source(x,&sol);
      cmasa_eval_t_an    (x,&an);
      printf("%g %g %g\n",x,sol,an);
     }

}//end program
```

# F90 Example: Euler Equations

```
program main
  using masa ! load masa module
  implicit none

  real(8) :: tempx,ufield,efield,rho,u_an,p_an,rho_an
  real(8) ::  lx =  1
  integer ::  i, error
  integer ::  nx = 10
  real(8) ::  dx = double(lx/nx)

  call masa_init("euler-example","euler_1d")
  call masa_init_param()
  call masa_sanity_check()

  do i=0,nx
      tempx = i*dx

      call masa_eval_u_source  (tempx,ufield,error)
      call masa_eval_e_source  (tempx,efield,error)
      call masa_eval_rho_source(tempx,rho   ,error)

      call masa_eval_u_an      (tempx,u_an  ,error)
      call masa_eval_p_an      (tempx,p_an  ,error)
      call masa_eval_rho_an    (tempx,rho_an,error)
  enddo

end program main
```

# C++ Example: Switching between solutions

```
#include <masa.h>
using namespace MASA;

int main()
{
  double solution;

  masa_init("alice","heateq_1d_steady_const");
  masa_init_param();

  masa_init("bob"  ,"euler_2d");
  masa_init_param();

  masa_select_mms("alice");
  masa_eval_t_source(1.2,&solution);
  cout << solution << endl;

  masa_select_mms("bob");
  masa_eval_u_source(1,1,&solution);
  cout << solution << endl;

}// end program
```

# Pseudocode/C example of using masa

```
int main()
{

  masa_init("eq1","masa_equation");
  masa_init_param();

  RHS += masa_eval_source();
  SOL  = solve(RHS);
  AN   = masa_eval_an();

  L2(AN,SOL);

}
```

# Who Verifies the Verifiers

## Project conforms to rigorous PECOS standards

- regression testing
  - verified against maple c-output
    - Residual less than 10e-15
    - All Source terms
    - All analytical terms
  - compare to maple output (still in progress)
  - automatic testing on buildbot
- subversion (svn) for source revision control
- code reviews among PECOS developers
- doxygen document generation, model documents, etc.

# Outline

1 Introduction to Verification

2 Manufactured Solutions

3 Code Design

4 Conclusions & Future Work

# Future Work

- Actual Field Testing: Rhys, Karl, Paul, etc.
- Axisymmetric Navier Stokes and Euler
- Shock Tube (Sod Euler Equations)
- Develop MMS for Chemistry problems (Juan, Marco)
- Radiation, etc.
- Autoimport – script
- Public (open source) release

## other ideas

- Stronger regression tests
- Other physical systems
- Altered API
- Additional supported languages?

Thank you!

Questions/Comments?

**Thanks go out to:**

| | |
|---|---|
| Kemelli | Karl Schulz |
| Paul Bauman | Chris Simmons |
| Roy Stogner | Robert Moser |