

# Puffin User Manual

---

November 25, 2003



Johan Hoffman and Anders Logg

This manual is written by:  
Johan Hoffman and Anders Logg.

## Contents

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Introduction</b>                            | <b>4</b>  |
| <b>2</b> | <b>Mesh format</b>                             | <b>5</b>  |
| <b>3</b> | <b>Specifying the variational formulation</b>  | <b>6</b>  |
| <b>4</b> | <b>The function AssembleMatrix()</b>           | <b>8</b>  |
| <b>5</b> | <b>The function AssembleVector()</b>           | <b>9</b>  |
| <b>6</b> | <b>Example programs</b>                        | <b>10</b> |
| 6.1      | The program PoissonSolver . . . . .            | 10        |
| 6.2      | The program ConvDiffStationarySolver . . . . . | 10        |
| 6.3      | The program ConvDiffTimeDepSolver . . . . .    | 10        |

# 1 Introduction

This is a first draft for a Puffin manual. Contributions are most welcome.

Puffin is a simple and minimal implementation of FEniCS for Octave (MATLAB). It is based around the two functions `AssembleMatrix()` and `AssembleVector()`, which are used to assemble a linear system

$$AU = b, \tag{1}$$

representing a variational formulation of a differential equation,

$$a(u, v; w) = l(v; w) \quad \forall v \in V, \tag{2}$$

where  $a(u, v; w)$  is a bilinear form in  $u$  (the trial function) and  $v$  (the test function), and  $l(v; w)$  is a linear form in  $v$ . The bilinear form  $a$  and the linear form  $l$  are allowed to depend on a vector  $w$  (the coefficients), and are referred to as the left-hand side and the right-hand side of the variational formulation.

## 2 Mesh format

The mesh is assumed to be given by the three matrices `points`, `edges`, and `triangles`, representing the *points*, *edges*, and *triangles* of an unstructured triangular mesh in the plane.

**FIXME:** Add detailed description of the format.

### 3 Specifying the variational formulation

A variational formulation is given as a function in the following way:

```
function integral = MyForm(u, v, w, du, dv, dw, dx, ds, x, d, t, eq)

if eq == 1
    integral = ... * dx + ... * ds;
else
    integral = ... * dx + ... * ds;
end
```

The output argument `integral` should be the integral of the left-hand or right-hand side (depending on the value of `eq`) of the variational formulation over a triangle (with area `dx`) or an edge (with length `ds`).

The input arguments are given by

- `u`: the value of the trial function  $u$ ,
- `v`: the value of the test function  $v$ ,
- `w`: a vector of coefficient values (optional),
- `du`: the gradient  $\nabla u$  of the trial function  $u$ ,
- `dv`: the gradient  $\nabla v$  of the test function  $v$ ,
- `dw`: a matrix of coefficient gradients (optional),
- `dx`: the area of the current triangle (zero if we are on an edge),
- `ds`: the length of the current edge (zero if we are on a triangle),
- `x`: the current quadrature point,
- `d`: the current domain number or edge number,
- `t`: the value of time,

- eq: specifies left-hand side (1) or right-hand side (2) of the variational formulation.

As an example, consider the specification of the variational formulation for Poisson's equation. The variational formulation in mathematical notation is given by

$$\int_{\Omega} \nabla u \cdot \nabla v \, dx + \int_{\Gamma} \gamma uv \, ds = \int_{\Omega} f v \, dx + \int_{\Gamma} (\gamma g_D - g_N) v \, ds \quad \forall v. \quad (3)$$

For a certain choice of  $a$ ,  $f$ ,  $\gamma$ ,  $g_D$ , and  $g_N$ , this can be specified by a function `Poisson()` as follows.

```
function integral = Poisson(u, v, w, du, dv, dw, dx, ds, x, d, t, eq)

if eq == 1
    integral = du'*dv*dx + g(x,d,t)*u*v*ds;
else
    integral = f(x,d,t)*v*dx + (g(x,d,t)*gd(x,d,t) - gn(x,d,t))*v*ds;
end

%--- Conductivity (penalty factor) ---
function y = g(x, d, t)
y = 1e7;

%--- Dirichlet boundary condition ----
function y = gd(x, d, t)
y = 0;

%--- Neumann boundary condition ---
function y = gn(x, d, t)
y = 0;

%--- Right-hand side, source term ---
function y = f(x, d, t)
y = 5*pi^2*sin(pi*x(1))*sin(2*pi*x(2));
```

## 4 The function `AssembleMatrix()`

The syntax of the function `AssembleMatrix()` is

```
A = AssembleMatrix(points, edges, triangles, pde, W, time),
```

where the output argument is `A`, the matrix to be assembled, and the input arguments are given by

- `points`: the matrix containing the node coordinates of the mesh,
- `edges`: the matrix containing the edges of the mesh,
- `triangles`: the matrix containing the triangles of the mesh,
- `pde`: the name of the function specifying the variational formulation,
- `W`: a matrix where each column contains the nodal values of additional functions (coefficients) appearing in the variational formulation as the functions `w(1)`, `w(2)` and so on (leave empty (`[]`) if not needed),
- `time`: the value of time (the variable `t` in the variational formulation).

## 5 The function AssembleVector()

The syntax of the function AssembleVector() is

```
b = AssembleVector(points, edges, triangles, pde, W, time),
```

where the output argument is `b`, the vector to be assembled, and the input arguments are given by

- `points`: the matrix containing the node coordinates of the mesh,
- `edges`: the matrix containing the edges of the mesh,
- `triangles`: the matrix containing the triangles of the mesh,
- `pde`: the name of the function specifying the variational formulation,
- `W`: a matrix where each column contains the nodal values of additional functions (coefficients) appearing in the variational formulation as the functions `w(1)`, `w(2)` and so on (leave empty (`[]`) if not needed),
- `time`: the value of time (the variable `t` in the variational formulation).

## 6 Example programs

### 6.1 The program `PoissonSolver`

The program `PoissonSolver` solves Poisson's equation on the unit square, using the variational formulation specified by the function `Poisson()`.

### 6.2 The program `ConvDiffStationarySolver`

The program `ConvDiffStationarySolver` solves the stationary convection-diffusion equation around a cylinder, using the variational formulation specified by the function `ConvDiffStationary()`.

### 6.3 The program `ConvDiffTimeDepSolver`

The program `ConvDiffTimeDepSolver` solves the time-dependent convection-diffusion equation on the unit square, using the variational formulation specified by the function `ConvDiffTimeDep()`.