generateCode

July 23, 2025

[1]:
```
# Code generation
# =======================
from casadi import *
```

Let's build a trivial symbolic SX graph

[2]:
```
x = SX.sym("x")
y = SX.sym("y")
z = x*y+2*y
z += 4*z
```

A Function is needed to inspect the graph

[3]:
```
f = Function("f", [x,y],[z])
```

The default representation is just the name of the function

[4]:
```
print(f.__repr__())
```

```
Function(f:(i0,i1)->(o0) SXFunction)
```

A print statement will call **str**() The result will look like a node-by-node tree evaluation

[5]:
```
print(f)
```

```
f:(i0,i1)->(o0) SXFunction
```

The generate method will insert this node-by-node evaluation in exported C code

[6]:
```
f.generate("f_generated")
```

[6]:
```
'f_generated.c'
```

This is how the exported code looks like:

[7]:
```
print(open('f_generated.c').read())
```

```
/* This file was automatically generated by CasADi 3.7.1.
 *  It consists of:
 *   1) content generated by CasADi runtime: not copyrighted
 *   2) template code copied from CasADi source: permissively licensed (MIT-0)
 *   3) user code: owned by the user
 *
 */
#ifdef __cplusplus
extern "C" {
#endif

/* How to prefix internal symbols */
#ifdef CASADI_CODEGEN_PREFIX
  #define CASADI_NAMESPACE_CONCAT(NS, ID) _CASADI_NAMESPACE_CONCAT(NS, ID)
  #define _CASADI_NAMESPACE_CONCAT(NS, ID) NS ## ID
  #define CASADI_PREFIX(ID) CASADI_NAMESPACE_CONCAT(CODEGEN_PREFIX, ID)
#else
  #define CASADI_PREFIX(ID) f_generated_ ## ID
#endif

#include <math.h>

#ifndef casadi_real
#define casadi_real double
#endif

#ifndef casadi_int
#define casadi_int long long int
#endif

/* Add prefix to internal symbols */
```

```
#define casadi_f0 CASADI_PREFIX(f0)
#define casadi_s0 CASADI_PREFIX(s0)

/* Symbol visibility in DLLs */
#ifndef CASADI_SYMBOL_EXPORT
  #if defined(_WIN32) || defined(__WIN32__) || defined(__CYGWIN__)
    #if defined(STATIC_LINKED)
      #define CASADI_SYMBOL_EXPORT
    #else
      #define CASADI_SYMBOL_EXPORT __declspec(dllexport)
    #endif
  #elif defined(__GNUC__) && defined(GCC_HASCLASSVISIBILITY)
    #define CASADI_SYMBOL_EXPORT __attribute__ ((visibility ("default")))
  #else
    #define CASADI_SYMBOL_EXPORT
  #endif
#endif

static const casadi_int casadi_s0[3] = {1, 1, 1};

/* f:(i0,i1)->(o0) */
static int casadi_f0(const casadi_real** arg, casadi_real** res, casadi_int* iw,
casadi_real* w, int mem) {
  casadi_real a0, a1, a2;
  a0=arg[0]? arg[0][0] : 0;
  a1=arg[1]? arg[1][0] : 0;
  a0=(a0*a1);
  a2=2.;
  a2=(a2*a1);
  a0=(a0+a2);
  a2=4.;
  a2=(a2*a0);
  a0=(a0+a2);
  if (res[0]!=0) res[0][0]=a0;
  return 0;
}

CASADI_SYMBOL_EXPORT int f(const casadi_real** arg, casadi_real** res,
casadi_int* iw, casadi_real* w, int mem){
  return casadi_f0(arg, res, iw, w, mem);
}

CASADI_SYMBOL_EXPORT int f_alloc_mem(void) {
  return 0;
}

CASADI_SYMBOL_EXPORT int f_init_mem(int mem) {
  return 0;
```

```
}

CASADI_SYMBOL_EXPORT void f_free_mem(int mem) {
}

CASADI_SYMBOL_EXPORT int f_checkout(void) {
  return 0;
}

CASADI_SYMBOL_EXPORT void f_release(int mem) {
}

CASADI_SYMBOL_EXPORT void f_incref(void) {
}

CASADI_SYMBOL_EXPORT void f_decref(void) {
}

CASADI_SYMBOL_EXPORT casadi_int f_n_in(void) { return 2;}

CASADI_SYMBOL_EXPORT casadi_int f_n_out(void) { return 1;}

CASADI_SYMBOL_EXPORT casadi_real f_default_in(casadi_int i) {
  switch (i) {
    default: return 0;
  }
}

CASADI_SYMBOL_EXPORT const char* f_name_in(casadi_int i) {
  switch (i) {
    case 0: return "i0";
    case 1: return "i1";
    default: return 0;
  }
}

CASADI_SYMBOL_EXPORT const char* f_name_out(casadi_int i) {
  switch (i) {
    case 0: return "o0";
    default: return 0;
  }
}

CASADI_SYMBOL_EXPORT const casadi_int* f_sparsity_in(casadi_int i) {
  switch (i) {
    case 0: return casadi_s0;
    case 1: return casadi_s0;
    default: return 0;
```

```
  }
}

CASADI_SYMBOL_EXPORT const casadi_int* f_sparsity_out(casadi_int i) {
  switch (i) {
    case 0: return casadi_s0;
    default: return 0;
  }
}

CASADI_SYMBOL_EXPORT int f_work(casadi_int *sz_arg, casadi_int* sz_res,
casadi_int *sz_iw, casadi_int *sz_w) {
  if (sz_arg) *sz_arg = 2;
  if (sz_res) *sz_res = 1;
  if (sz_iw) *sz_iw = 0;
  if (sz_w) *sz_w = 0;
  return 0;
}

CASADI_SYMBOL_EXPORT int f_work_bytes(casadi_int *sz_arg, casadi_int* sz_res,
casadi_int *sz_iw, casadi_int *sz_w) {
  if (sz_arg) *sz_arg = 2*sizeof(const casadi_real*);
  if (sz_res) *sz_res = 1*sizeof(casadi_real*);
  if (sz_iw) *sz_iw = 0*sizeof(casadi_int);
  if (sz_w) *sz_w = 0*sizeof(casadi_real);
  return 0;
}


#ifdef __cplusplus
} /* extern "C" */
#endif
```